高等院校信息技术规划教材

.NET Web 企业应用开发实战

黄锐军　编著

清华大学出版社

高等院校信息技术规划教材

.NET Web 企业应用开发实战

黄锐军　编著

清 华 大 学 出 版 社

北　京

内 容 简 介

本书通过企业项目开发案例深入介绍.NET Web企业应用开发高级技术，共包含4个综合实践项目。第一个项目是基于百度的天气预报查询程序，讲解数据的XML与JSON序列化问题以及客户端与网站服务器数据上传与下载的方法。第二个项目是基于Web的图片共享程序，讲解通过自定义协议客户端与服务器的交互问题来实现数据上传与下载。第三个项目是基于微信的成绩查询程序，讲解Web Service程序的技术方法以及微信公众号程序的开发技术。第四个项目是基于WCF的试题练习程序，讲解WCF服务器与客户端程序的开发技术。所有项目的服务器程序大都采用三层架构设计思想，客户端采用WPF窗体程序。

本书的特点是实践性强，由浅入深，循序渐进，每个项目都分解成若干节，每节以案例展示、技术要点、服务器程序、客户端程序、拓展训练的结构展开。

本书可以作为高等院校的教材，也可以作为相关技术人员学习.NET Web程序设计的参考书。

图书在版编目(CIP)数据

.NET Web企业应用开发实战/黄锐军编著. —北京：清华大学出版社，2017
(高等院校信息技术规划教材)
ISBN 978-7-302-46531-7

Ⅰ. ①N…　Ⅱ. ①黄…　Ⅲ. ①计算机网络－程序设计－高等学校－教材　Ⅳ. ①TP393.09

中国版本图书馆CIP数据核字(2017)第025620号

责任编辑：张　玥　战晓雷
封面设计：常雪影
责任校对：白　蕾
责任印制：何　芊

出版发行：清华大学出版社
网　址：http://www.tup.com.cn，http://www.wqbook.com
地　址：北京清华大学学研大厦A座　**邮　编**：100084
社 总 机：010-62770175　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：三河市春园印刷有限公司
经　销：全国新华书店
开　本：185mm×260mm　**印　张**：17　**字　数**：393千字
版　次：2017年5月第1版　**印　次**：2017年5月第1次印刷
印　数：1～2000
定　价：39.50元

产品编号：072803-01

前言 Foreword

目前国内有大量关于ASP.NET网站设计的高校教材，但是微软公司的.NET Web技术远远不限于ASP.NET网页设计，还有更高级别的Web Service及WCF等技术。在引进的国外教程中有一些涉及这方面的技术，但是这些书籍一般内容很杂，实践证明这些书籍教学效果欠佳，不适合用作高校的教材。针对我国高校的教学实际，作者编写了这本由企业项目驱动的教程。

本教程包含4个项目，第一个项目是基于百度的天气预报查询程序，讲解数据的XML与JSON序列化问题，同时讲解客户端与网站服务器数据上传与下载的方法。第二个项目是基于Web的图片共享程序，讲解通过自定义协议客户端与服务器的交互问题来实现数据上传与下载。第三个项目是基于微信的成绩查询程序，讲解Web Service程序的技术方法，同时讲解微信公众号程序的开发技术。第四个项目是基于WCF的试题练习程序，讲解WCF服务器与客户端程序的开发技术。这几个项目的客户端大都设计成WPF的窗体程序。

要学好一门编程技术，不应该拘泥于该技术的规范细节，而应该大量使用该技术来编写程序，在实践中学习与巩固基本知识，锻炼编程能力。本书的特点是实践性强，每节内容都采用案例展示、技术要点、服务器程序、客户端程序、拓展训练的结构展开。读者先通过"案例展示"了解要做什么，通过"技术要点"学习要用到什么技术，通过"服务器程序"与"客户端程序"学习程序的编写方法，最后通过"拓展训练"巩固与拓展所学习的知识。

每个项目都配有一些编程练习，有些练习项目是比较复杂的，需要读者花费比较长的时间来完成。

本教程可以作为高等院校的教材，建议总学时安排在60学时左右，其中讲授与上机实习的学时比例为1∶1左右。

本书是作者省级精品资源在线开放课程“DotNetWeb编程应用程序实践”的配套教材，课程网站http://www.dotnetweb.com.cn提供大量的教学资源，课程还开通了微信公众号DotNetWeb微课教学。

由于作者水平有限，书中难免有错误或不足之处，敬请广大读者批评指正。



微信公众号：DotNetWeb

作　者

2017年1月于深圳

目录 Contents

项目一



基于百度的天气预报程序

设计一个天气预报程序是很有价值的，问题是如何去找天气的数据。实际上目前很多网站都提供天气预报的实时数据，百度天气预报就是其中之一，可以从这些网站上获取任何一个城市的天气预报数据，把它整合到自己的程序中。图 1-1 就是一个 WPF (Windows Premtation Foundation) 的窗体程序，启动后可以选择任何一个省份的任何一个城市，就显示出该城市 4 天内的天气预报，数据来自百度天气预报网站。

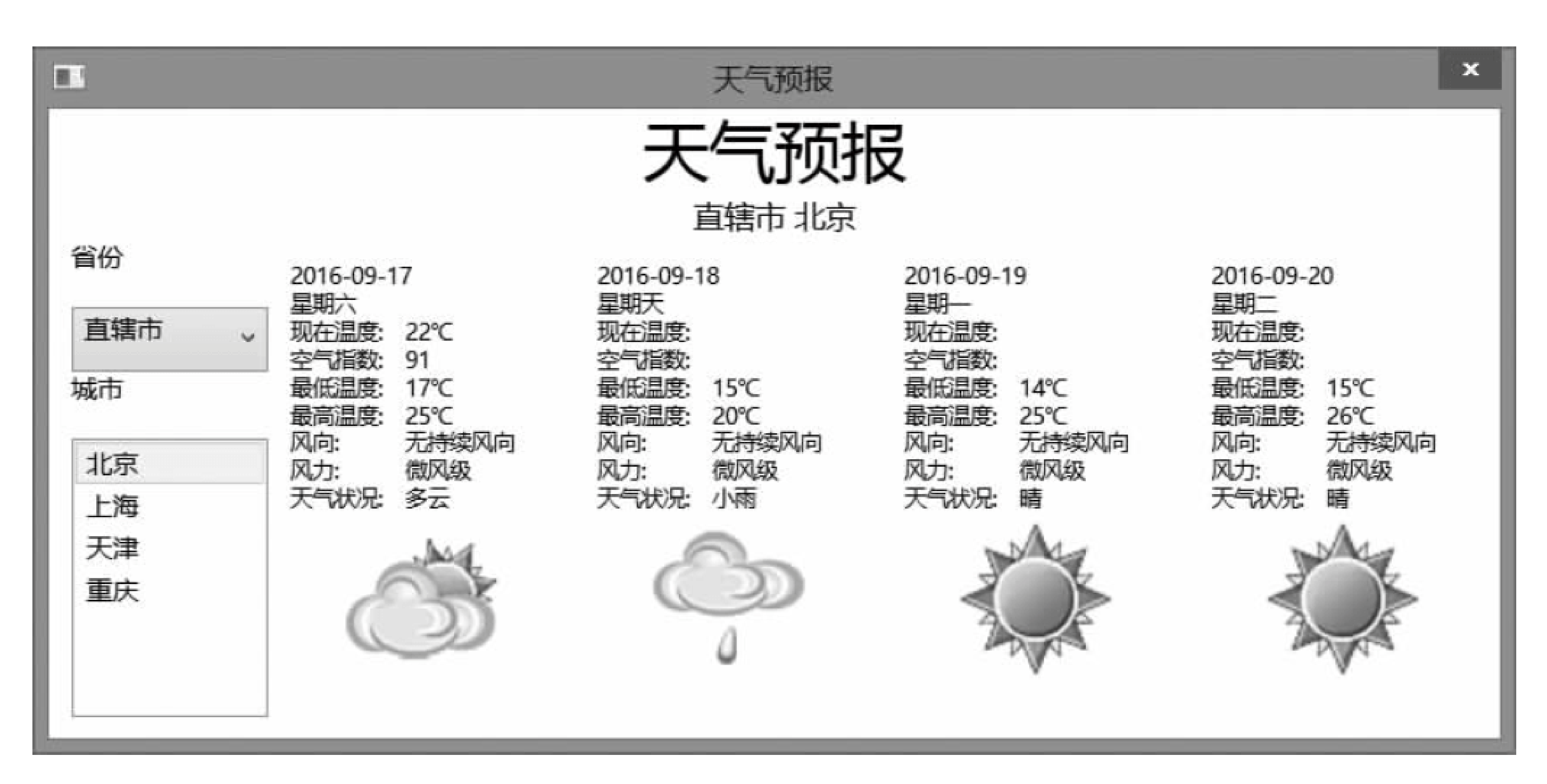


图 1-1　百度天气预报

这是一个典型的 C/S(Client/Server，客户/服务器)结构的程序，客户端是一个 WPF 的窗体程序，服务器是提供天气预报的百度网站。但百度是一个 Web 网站，给出的天气预报数据一般是 JSON 结构的，客户端要通过 HTTP 协议与这个网站进行交互，其程序结构如图 1-2 所示。

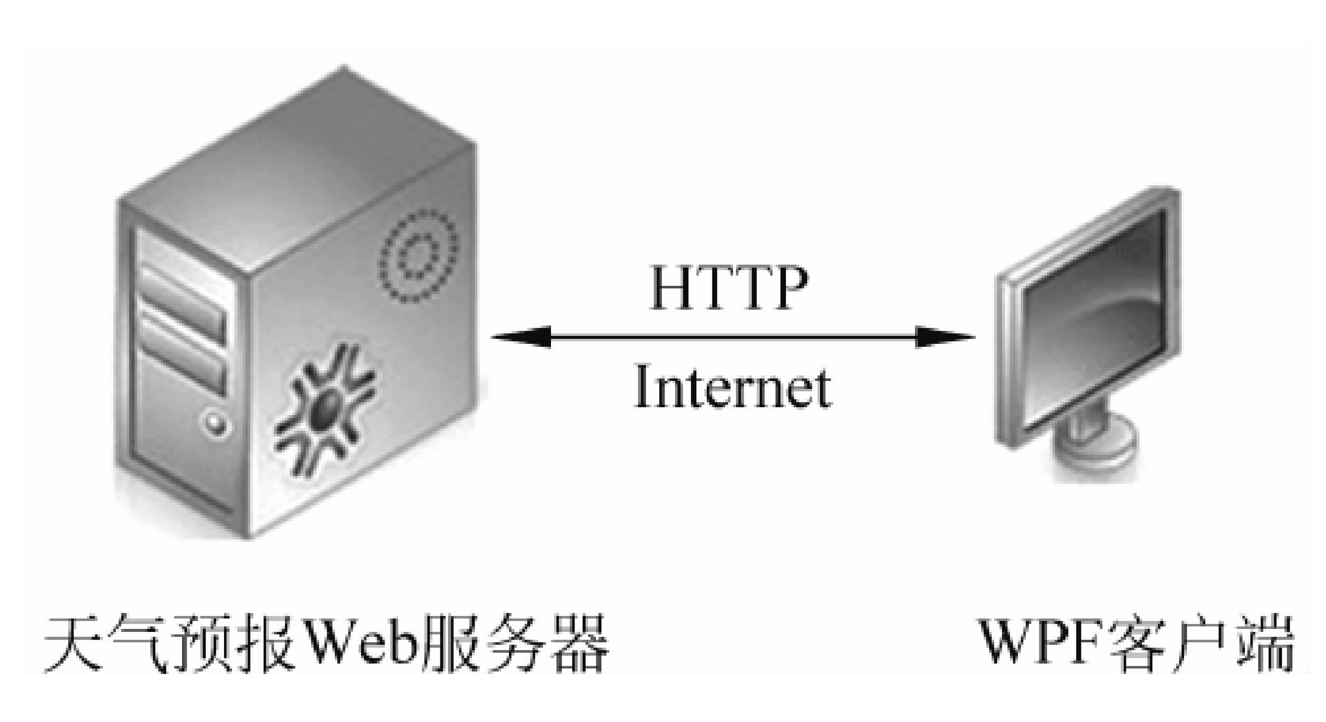


图 1-2　百度天气预报程序结构

1.1 服务器与客户端程序

1.1.1 案例展示

在 D:\web 目录下创建一个服务器程序 server. ashx，并把该目录设置为一个 Web 服务站点，通过浏览器访问地址 http://localhost/web/server. ashx 就可以看到输出的信息“Hello World 我的服务器”，如图 1-3 所示。

再设计一个 WPF 窗体的客户端程序，在客户端程序的地址栏中输入网址 http://localhost/web/server. ashx，单击“连接”按钮就可以看到服务器返回的信息“Hello World 我的服务器”，如图 1-4 所示。



图 1-3 浏览器访问服务器



图 1-4 窗体程序访问服务器

1.1.2 技术要点

客户端访问服务器的核心类是 WebClient 类，该类在 System. Net 空间中定义，该类可以用于客户端与 Web 服务器端进行通信，它有很多重要的方法，其中一个方法是

public String DownloadString(String url)

这个方法的作用是通过 GET 方法访问地址为 url 的网站，返回该网站的信息。例如：

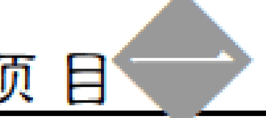
```
WebClient client=new WebClient();
client.Encoding=Encoding.UTF8;
String s=client.DownloadString("http://localhost/web/server.ashx");
```

程序首先创建一个 WebClient 对象 client，然后设置它的编码为因特网默认的 UTF8 编码，最后调用 DownloadString 方法访问网站，并返回网站的信息。

1.1.3 服务器程序

1. 创建网页

在 D 盘根目录建立 web 文件夹，用 Visual Studio 打开网站 D:\web，执行“网站”→“添加新项”命令，出现“添加新项”对话框，如图 1-5 所示。选择“一般处理程序”，这种程序的扩展名是 ashx，在“名称”中输入 server. ashx 后单击“添加”按钮，在网站中就增加了



一个程序 server.ashx。

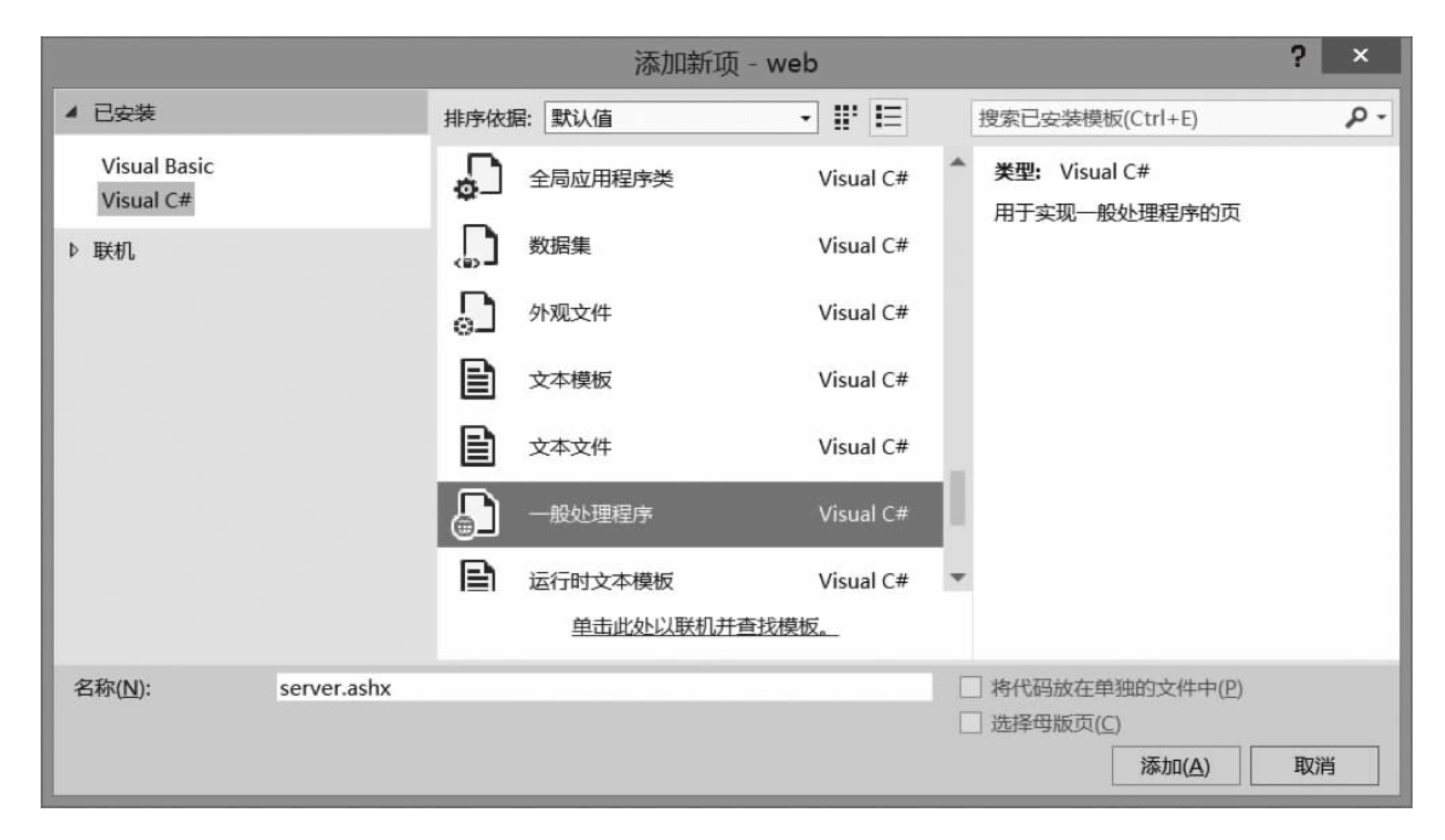


图 1-5 添加新项对话框

server.ashx 程序的代码如下：

```
<%@WebHandler Language="C# " Class="server" %>
using System;
using System.Web;
public class server : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType="text/plain";
        context.Response.Write("Hello World 我的服务器");
    }
    public bool IsReusable {
        get {
            return false;
        }
    }
}
```

其中 ProcessRequest 是主程序入口，context 对象是上下文对象，该对象包含了在网页设计中常见的 Request、Response 等对象，通过语句

```
context.Response.ContentType="text/plain";
```

设置要输出的文本的类型是纯文本，通过语句

```
context.Response.Write("Hello World 我的服务器");
```

输出“Hello World 我的服务器”。

2. 创建网站

上面建立的服务器程序 server.ashx 可以通过 Visual Studio 直接运行，但是程序调试好后要放在一个网站下独立运行，这需要把 D:\web 目录设置为网站，操作步骤如下：

(1) 打开“控制面板”的“管理工具”，选择“Internet Information Services (IIS)管理器”，弹出对话框。

(2) 默认有一个 Default Web Site 的网站，选择该网站，并右击，弹出快捷菜单，选择“添加应用程序”命令，弹出“添加应用程序”对话框，如图 1-6 所示。在对话框中选择“物理路径”为 D:\web，别名可以设置为 web(也可以是其他名字)，单击“确定”按钮后关闭对话框。



图 1-6 “添加应用程序”对话框

(3) 这时可以看到在 Default Web Site 下增加了一个名为 web 的项目，该项目就是默认网站下的一个站点，如图 1-7 所示。

在浏览器地址栏中输入网址 http://localhost/web/server.ashx 就可以看到图 1-3 所示的结果。在浏览器中查看网页源代码，可以看到服务器输出的只有一句“Hello World 我的服务器”的纯文本，没有别的 HTML 格式信息，如图 1-8 所示。



图 1-7 增加 web 站点

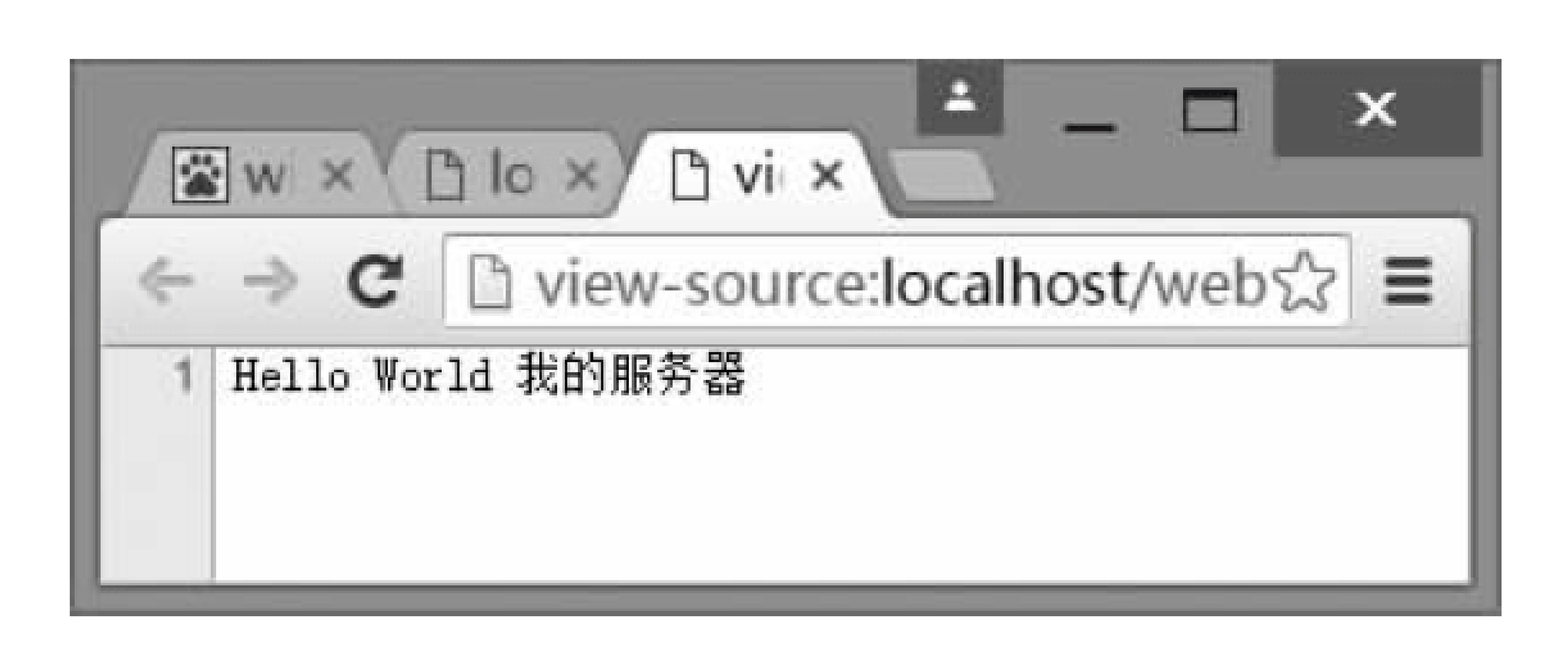


图 1-8 在浏览器中查看网页源代码

1.1.4 客户端程序

1. 界面设计

本项目把客户端程序设计成一个 WPF 程序，在窗体 MainWindow 中放入一个名为 txtUrl 的 TextBox 存放地址、一个名为 btCon 的 Button 连接按钮和一个名为 msg 显示标签 TextBlock，MainWindow. xaml 代码如下：

```
<Grid>
  <StackPanel Orientation="Vertical">
     <TextBox x:Name="txtUrl" Text="http://localhost/web/server.ashx" />
     <Button x:Name="btCon" Content="连接" Width="40" Click="btCon_Click" />
     <TextBlock x:Name="msg" />
  </StackPanel>
</Grid>
```

2. 程序设计

相应的 MainWindow. xaml. cs 程序如下：

```
namespace client
{
    public partial class MainWindow : Window
    {
        WebClient client;
        public MainWindow()
        {
            InitializeComponent();
        }
        private void btCon_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                WebClient client=new WebClient();
                client.Encoding=Encoding.UTF8;
                Uri uri=new Uri(txtUrl.Text, UriKind.Absolute);
                msg.Text=client.DownloadString(uri);
            }
            catch(Exception exp) { msg.Text=exp.Message; }
        }
    }
}
```

运行该程序，就得到图 1-4 所示的结果。

1.1.5 拓展训练

必须指出的是，WebClient 类的 DownloadString 方法在执行时会阻塞程序，等待服务器的响应，如果服务器长期没有响应，该方法就会一直等待下去，直到等待时间超出了最大的等待时间就抛出一个异常。这种等待过程称为同步过程，同步过程会阻塞程序，在网络程序中，除非能够确保服务器响应是很及时的，一般不建议采用同步过程，而是采用异步过程。

WebClient 提供了异步访问方法 DownloadStringAsync，声明如下：

public void DownloadStringAsync(String url)

该方法与同步方法 DownloadString 最大的不同就在于它不返回任何信息，在执行后不用等待服务器响应就立即返回，因此不会阻塞程序。该方法执行时只是确定了要访问指定的服务器，服务器什么时候响应是不可以预计的，它不负责等待服务器的返回。

但是一旦服务器响应，客户端又怎么样得到服务器响应的结果呢？实际上在异步方法执行前要先设置好一个回调函数，服务器响应后就触发该回调函数执行，获取服务器

响应结果。回调函数的设置如下：

```
void client_DownloadStringCompleted
(object sender, DownloadStringCompletedEventArgs e)
{
}
```

其中 client_DownloadStringCompleted 是回调函数名称，参数 e 的类型是 DownloadStringCompletedEventArgs。在服务器响应后就会触发该执行该函数，并把参数 e 传递给该函数，通过 e.Result 就得到服务器响应的信息。采用异步方法的程序如下：

```
private void btCon_Click(object sender, RoutedEventArgs e)
{
    try
    {
        WebClient client=new WebClient();
        client.Encoding=Encoding.UTF8;
        client.DownloadStringCompleted+=client_DownloadStringCompleted;
        Uri uri=new Uri(txtUrl.Text, UriKind.Absolute);
        client.DownloadStringAsync(uri);
        msg.Text="等待服务器响应...";
    }
    catch(Exception exp) { msg.Text=exp.Message; }
}
void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
{
    msg.Text=e.Result;
}
```

运行时单击“连接”按钮可以立即看到标签上显示“等待服务器响应...”，过一会服务器返回信息，触发 client_DownloadStringCompleted 函数执行，就看到服务器返回的信息“Hello World 我的服务器”。由此可见，执行 client.DownloadStringAsync(uri)时并不阻塞程序以等待服务器相应，而是立即执行 msg.Text = "等待服务器响应..."语句，待服务器响应后就触发 client_DownloadStringCompleted 函数，其中 e.Result 为服务器的返回信息。

一般为了监测服务器的错误，回调函数可以写成如下形式：

```
void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
{
    if(e.Eorr==null) msg.Text=e.Result;
    else showMsg(e.Error.Message);
}
```

如果没有错误，e. Error 就为 null，否则 e. Error 不为 null，而且 e. Error. Message 为错误信息。

1.2 XML 数据的网络传输

1.2.1 案例展示

服务器程序与客户端程序建立连接后要相互传递信息，双方要事先约定好这些信息的格式。XML 数据格式是常用的一种数据格式，如图 1-9 所示，服务器返回一个天气的 XML 字符串给客户端，客户端解析出每个字段的值。



图 1-9　XML 序列化

1.2.2 技术要点

数据在网络上传递必须是串行化、序列化的，因此数据在发送之前需要序列化。XML 序列化是一种常用的序列化方法，是把一个对象数据变成一个 XML 字符串的过程，可以采用 XmlSerializer 类对象完成这个过程。一个序列化的 XML 字符串经过网络传输到达另外一端后，要用反序列化的方法把它转换为对象类型。

例如，一个城市的天气数据包括城市名称 city、日期 date、天气描述 description、最高温度 highTemp、最低温度 lowTemp 等，可以定义两个天气类 WeatherItem 与 WeatherClass：

```
public class WeatherItemClass
{
    public String description { get; set; }
    public double highTemp { get; set; }
    public double lowTemp { get; set; }
}
public class WeatherClass
```

```
{
    public String city { get; set; }
    public String date { get; set; }
    public WeatherItemClass weather { get; set; }
}
```

序列化就是把一个 WeatherClass 对象变成一个 XML 字符串，反序列化就是把 XML 字符串再变回一个 WeatherClass 对象。

1. XML 序列化

可以编写序列化函数如下：

```
String serialize(WeatherClass w)
{
    XmlSerializer xml=new XmlSerializer(typeof(WeatherClass));
    MemoryStream ms=new MemoryStream();
    xml.Serialize(ms, w);
    String s=Encoding.UTF8.GetString(ms.ToArray());
    return s;
}
```

首先创建一个 XmlSerializer 对象：

```
XmlSerializer xml=new XmlSerializer(typeof(WeatherClass));
```

然后创建一个内存流：

```
MemoryStream ms=new MemoryStream();
```

通过 XmlSerializer 的 Serialize 方法把对象序列化到这个流中：

```
xml.Serialize(ms, obj);
```

最后再通过

```
String s=Encoding.UTF8.GetString(ms.ToArray());
```

把这个流转为 XML 字符串。

例如：

```
WeatherClass wa=new WeatherClass
{
    city="深圳",
    date="2016-09-08",
    weather=new WeatherItemClass
    { description="多云", lowTemp=25, highTemp=30 }
};
```

```
String s=serialize (w);
```

序列化后结果为

```
<?xml version="1.0"?>
<WeatherClass xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:
              xsd="http://www.w3.org/2001/XMLSchema">
  <city>深圳</city>
  <date>2016-09-08</date>
  <weather>
    <description>多云</description>
    <highTemp>30</highTemp>
    <lowTemp>25</lowTemp>
  </weather>
</WeatherClass>
```

2. XML 反序列化

可以编写反序列化函数如下：

```
WeatherClass deserialize(String s)
{
    XmlSerializer xml=new XmlSerializer(typeof(WeatherClass));
    byte[] buf=Encoding.UTF8.GetBytes(s);
    MemoryStream ms=new MemoryStream(buf);
    WeatherClass w= (WeatherClass)xml.Deserialize(ms);
    return w;
}
```

首先创建一个 XmlSerializer 对象：

```
XmlSerializer xml=new XmlSerializer(typeof(WeatherClass));
```

然后根据 XML 字符串 s 创建一个内存流：

```
byte[] buf=Encoding.UTF8.GetBytes(s);
MemoryStream ms=new MemoryStream(buf);
```

通过 XmlSerializer 的 Deserialize 方法把对象字符串流转回到 WeatherClass 对象：

```
WeatherClass w= (WeatherClass)xml.Deserialize(ms);
```

例如，上面的 XML 字符串反序列化后又回到 WeatherClass 类的对象。

1.2.3 服务器程序

服务器程序定义一个 WeatherItemClass 与 WeatherClass 类，把 WeatherClass 类的一个对象序列化成一个 XML 字符串，然后把这个字符串传递给客户端。

```
public class server : IHttpHandler
{
    public class WeatherItemClass
    {
        public String description { get; set; }
        public double highTemp { get; set; }
        public double lowTemp { get; set; }
        public override String ToString()
        {
            return description+","+lowTemp.ToString()+","+highTemp.ToString();
        }
    }
    public class WeatherClass
    {
        public String city { get; set; }
        public String date { get; set; }
        public WeatherItemClass weather { get; set; }
        public override String ToString()
        {
            return city+","+date+","+weather.ToString();
        }
    }
    String serialize(WeatherClass w)
    {
        XmlSerializer xml=new XmlSerializer(typeof(WeatherClass));
        MemoryStream ms=new MemoryStream();
        xml.Serialize(ms, w);
        String s=Encoding.UTF8.GetString(ms.ToArray());
        return s;
    }
    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType="text/plain";
        WeatherClass w=new WeatherClass {
            city="深圳", date="2016-09-08",
            weather=new WeatherItemClass
            { description="多云<晴天>", lowTemp=26, highTemp=30 }
        };
        String s=serialize(w);
        context.Response.Write(s);
    }
    public bool IsReusable {
        get {
            return false;
```

```
        }
    }
}
```

在浏览器中执行该服务器程序,可以看到输出结果:

```
<?xml version="1.0"?>
<WeatherClass xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:
               xsd="http://www.w3.org/2001/XMLSchema">
  <city>深圳</city>
  <date>2016-09-08</date>
  <weather>
    <description>多云 &lt;晴天 &gt;</description>
    <highTemp>30</highTemp>
    <lowTemp>26</lowTemp>
  </weather>
</WeatherClass>
```

1.2.4 客户端程序

1. 界面设计

客户端主要有一个名称为 tCon 的按钮 Button 用来连接服务器,一个名称为 txt 的文本框用来显示执行结果。

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="40" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Button x:Name="btCon"  Content="连接" Width="60" Height="30" Click=
            "btCon_Click"  />
    <TextBox x:Name="txt" Grid.Row="1" AcceptsReturn="True" IsReadOnly=
             "True" VerticalScrollBarVisibility="Auto" />
</Grid>
```

2. 程序设计

客户端也定义相同的 WeatherItemClass 与 WeatherClass 类,连接服务器获取 XML 字符串后,把这个 XML 字符串反序列化成 WeatherClass 对象。

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
```

```
        InitializeComponent();
    }
    public class WeatherItemClass
    {
        public String description { get; set; }
        public double highTemp { get; set; }
        public double lowTemp { get; set; }
        public override String ToString()
        {
            return description+","+lowTemp.ToString()+","+highTemp.ToString();
        }
    }
    public class WeatherClass
    {
        public String city { get; set; }
        public String date { get; set; }
        public WeatherItemClass weather { get; set; }
        public override String ToString()
        {
            return city+","+date+","+weather.ToString();
        }
    }
    WeatherClass deserialize(String s)
    {
        XmlSerializer xml=new XmlSerializer(typeof(WeatherClass));
        byte[] buf=Encoding.UTF8.GetBytes(s);
        MemoryStream ms=new MemoryStream(buf);
        WeatherClass w=(WeatherClass)xml.Deserialize(ms);
        return w;
    }
    void showMsg(String s)
    {
        txt.AppendText(s+"\r\n");
    }
    private void btCon_Click(object sender, RoutedEventArgs e)
    {
        try
        {
            WebClient client=new WebClient();
            client.Encoding=Encoding.UTF8;
            client.DownloadStringCompleted+=client_DownloadStringCompleted;
            client.DownloadStringAsync(new Uri("http://localhost/web/server.
            ashx"));
        }
```

```
            catch(Exception exp) { showMsg(exp.Message); }
        }
        void client_DownloadStringCompleted(object sender,
        DownloadStringCompletedEventArgs e)
        {
            try
            {
                WeatherClass w=deserialize(e.Result);
                showMsg(e.Result);
                showMsg("\r\n反序列化对象: "+w.ToString());
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
```

执行该程序，结果如图 1-9 所示。

1.2.5 拓展训练

通常要序列化的对象多种多样，不能为每种对象都编写一个序列化与反序列化函数。实际上可以采用C#的泛型，就可以编写出适用于任何类型的序列化与反序列化函数，采用<T>泛型编写的函数如下：

```
String serialize<T>(T obj)
{
    XmlSerializer xml=new XmlSerializer(typeof(T));
    MemoryStream ms=new MemoryStream();
    xml.Serialize(ms, obj);
    String s=Encoding.UTF8.GetString(ms.ToArray());
    return s;
}
T deserialize<T>(String s)
{
    XmlSerializer xml=new XmlSerializer(typeof(T));
    byte[] buf=Encoding.UTF8.GetBytes(s);
    MemoryStream ms=new MemoryStream(buf);
    T obj=(T)xml.Deserialize(ms);
    return obj;
}
```

其中 T 代表任何的数据类型，采用这样的泛型函数后，序列化语句改为

```
String s=serialize<WeatherClass>(w);
```

反序列化语句改为

```
WeatherClass w=deserialize<WeatherClass>(s);
```

执行的结果完全一样。

1.3 JSON 数据的网络传输

1.3.1 案例展示

服务器程序与客户端程序建立连接后要相互传递信息，双方要事先约定好这些信息的格式。JSON 数据是一种常用的数据格式，如图 1-10 所示，服务器返回一个 JSON 格式的字符串描述城市的天气，客户端再把它反序列化，解析出各个字段的值。



图 1-10 JSON 序列化

1.3.2 技术要点

除了 XML 序列化外，JSON 序列化也是一种常用的序列化方法。JSON 序列化就是把一个对象数据变成一个 JSON 字符串的过程，可以采用 DataContractJsonSerializer 类对象完成这个过程。反过来，一个序列化的 JSON 字符串要用反序列化的方法把它转换为对象类型。

例如，一个城市的天气数据包括城市名称 city、日期 date、天气描述 description、最高温度 highTemp、最低温度 lowTemp 等，可以定义一个天气类 WeatherClass 来描述：

```
public class WeatherItemClass
{
    public String description { get; set; }
    public double highTemp { get; set; }
    public double lowTemp { get; set; }
}
public class WeatherClass
{
    public String city { get; set; }
    public String date { get; set; }
    public WeatherItemClass weather { get; set; }
}
```

类似于 XML 序列化函数，可以编写通用的 JSON 序列化函数 Serialize 与反序列化

函数 Deserialize 如下：

```
String serialize<T>(T obj)
{
    DataContractJsonSerializer json=new DataContractJsonSerializer(typeof(T));
    MemoryStream ms=new MemoryStream();
    json.WriteObject(ms, obj);
    String s=Encoding.UTF8.GetString(ms.ToArray());
    return s;
}
T deserialize<T>(String s)
{
    DataContractJsonSerializer json=new DataContractJsonSerializer(typeof(T));
    byte[] buf=Encoding.UTF8.GetBytes(s);
    MemoryStream ms=new MemoryStream(buf);
    T obj=(T)json.ReadObject(ms);
    return obj;
}
```

不同的是此处使用 DataContractJsonSerializer 类，这个类在应用时要先引入。引入的方法是执行菜单"项目"→"添加引用"命令，打开"引用管理器"对话框，如图 1-11 所示，选择引用 System. Runtime. Serialization。



图 1-11 引用 System. Runtime. Serialization

1.3.3 服务器程序

服务器程序定义 WeatherItemClass 与 WeatherClass 类，把 WeatherClass 类的一个对象序列化成一个 JSON 字符串，然后把这个字符串传递给客户端。

```
public class server : IHttpHandler
{
    public class WeatherItemClass
    {
        public String description { get; set; }
        public double highTemp { get; set; }
        public double lowTemp { get; set; }
        public override String ToString()
        {
            return description+","+lowTemp.ToString()+","+highTemp.ToString();
        }
    }
    public class WeatherClass
    {
        public String city { get; set; }
        public String date { get; set; }
        public WeatherItemClass weather { get; set; }
        public override String ToString()
        {
            return city+","+date+","+weather.ToString();
        }
    }
    String serialize<T>(T obj)
    {
        DataContractJsonSerializer json=new DataContractJsonSerializer(typeof(T));
        MemoryStream ms=new MemoryStream();
        json.WriteObject(ms, obj);
        String s=Encoding.UTF8.GetString(ms.ToArray());
        return s;
    }
    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType="text/plain";
        WeatherClass w=new WeatherClass
        {
            city="深圳",
            date="2016-09-08",
            weather=new WeatherItemClass
            { description="多云<晴天>", lowTemp=26, highTemp=30 }
        };
        String s=serialize(w);
        context.Response.Write(s);
    }
    public bool IsReusable {
```

```
        get {
            return false;
        }
    }
}
```

1.3.4 客户端程序

1. 界面设计

客户端主要有一个名称为 tCon 的按钮 Button 用来连接服务器，一个名称为 txt 的文本框用来显示执行结果。

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="40" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Button x:Name="btCon"  Content="连接" Width="60" Height="30" Click=
            "btCon_Click"  />
    <TextBox x:Name="txt" Grid.Row="1" AcceptsReturn="True" IsReadOnly=
             "True" VerticalScrollBarVisibility="Auto" />
</Grid>
```

2. 程序设计

客户端也定义相同的 WeatherItemClass 与 WeatherClass 类，连接服务器获取 JSON 字符串后，把这个 JSON 字符串反序列化成 WeatherClass 对象。

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }
    public class WeatherItemClass
    {
        public String description { get; set; }
        public double highTemp { get; set; }
        public double lowTemp { get; set; }
        public override String ToString()
        {
            return description +"," + lowTemp.ToString() +"," + highTemp.
            ToString();
        }
```

```
}
public class WeatherClass
{
    public String city { get; set; }
    public String date { get; set; }
    public WeatherItemClass weather { get; set; }
    public override String ToString()
    {
        return city+","+date+","+weather.ToString();
    }
}
T deserialize<T>(String s)
{
    DataContractJsonSerializer json=new DataContractJsonSerializer(typeof(T));
    byte[] buf=Encoding.UTF8.GetBytes(s);
    MemoryStream ms=new MemoryStream(buf);
    T obj=(T)json.ReadObject(ms);
    return obj;
}
void showMsg(String s)
{
    txt.AppendText(s+"\r\n");
}
private void btCon_Click(object sender, RoutedEventArgs e)
{
    try
    {
        WebClient client=new WebClient();
        client.Encoding=Encoding.UTF8;
        client.DownloadStringCompleted+=client_DownloadString-
        Completed;
        client.DownloadStringAsync(new Uri("http://localhost/web/server.
        ashx"));
    }
    catch(Exception exp) { showMsg(exp.Message); }
}
void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
{
    try
    {
        WeatherClass w=deserialize<WeatherClass>(e.Result);
        showMsg(e.Result);
        showMsg("\r\n 反序列化对象："+w.ToString());
    }
```

```
        catch(Exception exp) { showMsg(exp.Message); }
    }
}
```

1.3.5 拓展训练

XML序列化与JSON序列化都是常用的序列化方法，它们生成的字符串格式不一样。一般XML格式是计算机与人都比较容易读的字符串，而JSON字符串是计算机易读而人不易读的，同样是序列化一个对象，一般XML序列化得到的字符串会比JSON的长一些。

在处理二进制数据时XML序列化与JSON序列化的处理方法有点不同，XML序列化时会把二进制数据转为Base64的字符串，而JSON一般把二进制数据转为一个字符串数组，例如：

```
byte[] data=new byte[] {1,2,3,4,5};
```

如果采用XML序列化，那么转成的XML字符串为

```
<?xml version="1.0"?><base64Binary>AQIDBAU=</base64Binary>
```

其中的AQIDBAU为这个二进制数据的Base64字符串。

但是JSON序列化时转成的JSON字符串是

```
[1,2,3,4,5]
```

由此可见，JSON的转换比较简单直接。

1.4 城市天气状况查询

1.4.1 案例展示

在百度API网站中有全国各地的天气预报数据，设计一个程序从该网站获取并显示一个城市当天的天气状况数据，如图1-12所示。



图1-12 城市天气状况

1.4.2 技术要点

1. 百度 API Store 天气预报

很多网站提供天气预报数据，其中百度 API Store 就有全国各地的天气预报数据，有些是要付费的，也有免费的。要获取免费的天气预报数据，可以进入百度 API Store 的网站 http://apistore.baidu.com/，选择"生活常用"→"免费"→"天气查询"，如图 1-13 所示。



图 1-13 百度天气查询

查询城市天气，首先要申请一个 apikey，这是百度分配给用户的一个 ID 号。单击"获取 apikey"，弹出登录百度账号的对话框，如果没有百度账号，可以注册一个，登录成功后再次单击"获取 apikey"就可以得到一个字符串。

每个百度账户的 apikey 是不同的，而且百度建议不要把自己的 apikey 给别人使用，因此可以把自己的 apikey 存储在项目的资源文件中。例如，目前的程序项目是 client，执行 Visual Studio 的"项目"菜单中的"client 属性"命令，打开 client 项目的资源，在名称中输入 apikey，在值中输入 apikey 值，然后保存，如图 1-14 所示。



图 1-14 存储 apikey

程序存储的 apikey 可以通过下列语句获取：

```
Properties.Resources.apikey
```

其中 Properties. Resources 是项目中的资源，这个资源中有一个名为 apikey 的项目，通过

它获取apikey的值。

2. 获取天气数据

根据百度天气预报的说明,可以访问网址

```
http://apis.baidu.com/apistore/weatherservice/cityname
```

要查询某个城市的天气预报,只要在网址的后面加上"cityname=城市"的参数,例如:

```
http://apis.baidu.com/apistore/weatherservice/cityname?cityname=深圳
```

就可以查询到深圳当天的天气数据。

但是在访问网站之前必须在HTTP协议的头(Headers)中设置apikey,例如:

```
WebClient client=new WebClient();
client.Encoding=Encoding.UTF8;
client.Headers["apikey"]=Properties.Resources.apikey;
client.DownloadStringCompleted+=client_DownloadStringCompleted;
client. DownloadStringAsync (new Uri (" http://apis. baidu. com/apistore/
weatherservice/cityname?cityname=深圳",UriKind.Absolute));
```

在访问天气预报的网站时会把apikey传递给服务器,服务器用这个apikey对用户进行验证。

3. 解析天气数据

根据百度天气预报的说明,返回的是JSON数据。例如,北京的天气预报数据如下:

```
{
    errNum: 0,
    errMsg: "success",
    retData: {
        city: "北京",                 //城市
        pinyin: "beijing",            //城市拼音
        citycode: "101010100",        //城市编码
        date: "15-02-11",             //日期
        time: "11:00",                //发布时间
        postCode: "100000",           //邮编
        longitude: 116.391,           //经度
        latitude: 39.904,             //维度
        altitude: "33",               //海拔
        weather: "晴",                //天气情况
        temp: "10",                   //气温
        l_tmp: "-4",                  //最低气温
        h_tmp: "10",                  //最高气温
        WD: "无持续风向",             //风向
```

```
        WS: "微风(<10m/h)",            //风力
        sunrise: "07:12",              //日出时间
        sunset: "17:44"                //日落时间
    }
}
```

其中,errNum 是错误指示,如果为 0 表示没有错误;errMsg 是错误信息,没有错误时为 success;retData 是返回的数据对象,这个对象中包含很多天气信息,例如,weather 为天气状况,l_tmp 与 h_tmp 为最低温度与最高温度等。

我们不一定关心所有的天气数据,根据自己需要的信息可以设计一个类与该数据对应。例如,我们只关心天气状况 weather、当前温度 temp、最低与最高温度 l_tmp 与 h_tmp、风向 WD、风力 WS,那么可以设计天气类 WeatherData 如下:

```
public class WeatherData
{
    public String city { get; set; }
    public String date { get; set; }
    public String time { get; set; }
    public String weather { get; set; }
    public String temp { get; set; }
    public String l_tmp { get; set; }
    public String h_tmp { get; set; }
    public String WD { get; set; }
    public String WS { get; set; }
}
public class WeatherClass
{
    public int errNum { get; set; }
    public String errMsg { get; set; }
    public WeatherData retData { get; set; }
}
```

执行 Visual Studio 的“项目”→“添加类”命令,把这些类存储在 WeatherClass.cs 文件中。

1.4.3 服务器程序

服务器程序是百度天气预报网站 http://apis.baidu.com/apistore/weatherservice/cityname。

1.4.4 客户端程序

1. 界面设计

这个程序界面很简单,主要是一个名为 btCon 的 Button 按钮和一个名为 txt 的 TextBox 文本框。

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="40" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Button x:Name="btCon"  Content="查询" Width="60" Height="30" Click=
        "btCon_Click"  />
    <TextBox x:Name="txt" Grid.Row="1" AcceptsReturn="True" IsReadOnly=
        "True" VerticalScrollBarVisibility="Auto" />
</Grid>
```

2. 程序设计

客户端程序按要求访问服务器,同时提供 apikey,服务器返回的 JSON 字符串反序列化为 WeatherClass 类对象,这个类对象中的 retData 是一个 WeatherData 对象,从这个对象就得到各种天气数据。

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }
    T deserialize<T>(String s)
    {
        DataContractJsonSerializer json=new DataContractJsonSerializer(typeof(T));
        byte[] buf=Encoding.UTF8.GetBytes(s);
        MemoryStream ms=new MemoryStream(buf);
        T obj=(T)json.ReadObject(ms);
        return obj;
    }
    void showMsg(String s)
    {
        txt.AppendText(s+"\r\n");
    }
    void client_DownloadStringCompleted(object sender,
    DownloadStringCompletedEventArgs e)
    {
        try
        {
            WeatherClass w=deserialize<WeatherClass>(e.Result);
            if(w.errNum==0)
            {
                String s="城市: "+w.retData.city+"\r\n";
                s=s+"日期: "+w.retData.date+"\r\n";
                s=s+"时间: "+w.retData.time+"\r\n";
                s=s+"当前温度: "+w.retData.temp+"\r\n";
```

```
                s=s+"最低温度："+w.retData.l_tmp+"\r\n";
                s=s+"最高温度："+w.retData.h_tmp+"\r\n";
                s=s+"天气状态："+w.retData.weather+"\r\n";
                s=s+"风向："+w.retData.WD+"\r\n";
                s=s+"风力："+w.retData.WS+"\r\n";
                showMsg(s);
            }
            else showMsg(w.errMsg);
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
    private void btCon_Click(object sender, RoutedEventArgs e)
    {
        try
        {
            WebClient client=new WebClient();
            client.Encoding=Encoding.UTF8;
            client.Headers["apikey"]=Properties.Resources.apikey;
            client.DownloadStringCompleted+=client_DownloadStringCompleted;
            client. DownloadStringAsync (new Uri ( " http://apis. baidu. com/
            apistore/weatherservice/cityname?cityname=深圳", UriKind.Absolute));
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
}
```

运行这个程序就可以看到该城市当天的天气状况。

1.4.5 拓展训练

可以改造这个程序，引入一个名称为 cbCity 的 ComboBox 控件来存储几个城市，当选择一个城市时就查询到该城市的天气信息，如图 1-15 所示。由于程序中要反复用到 WebClient 对象查询不同城市的天气信息，因此把 WebClient 对象 client 设置为类对象。

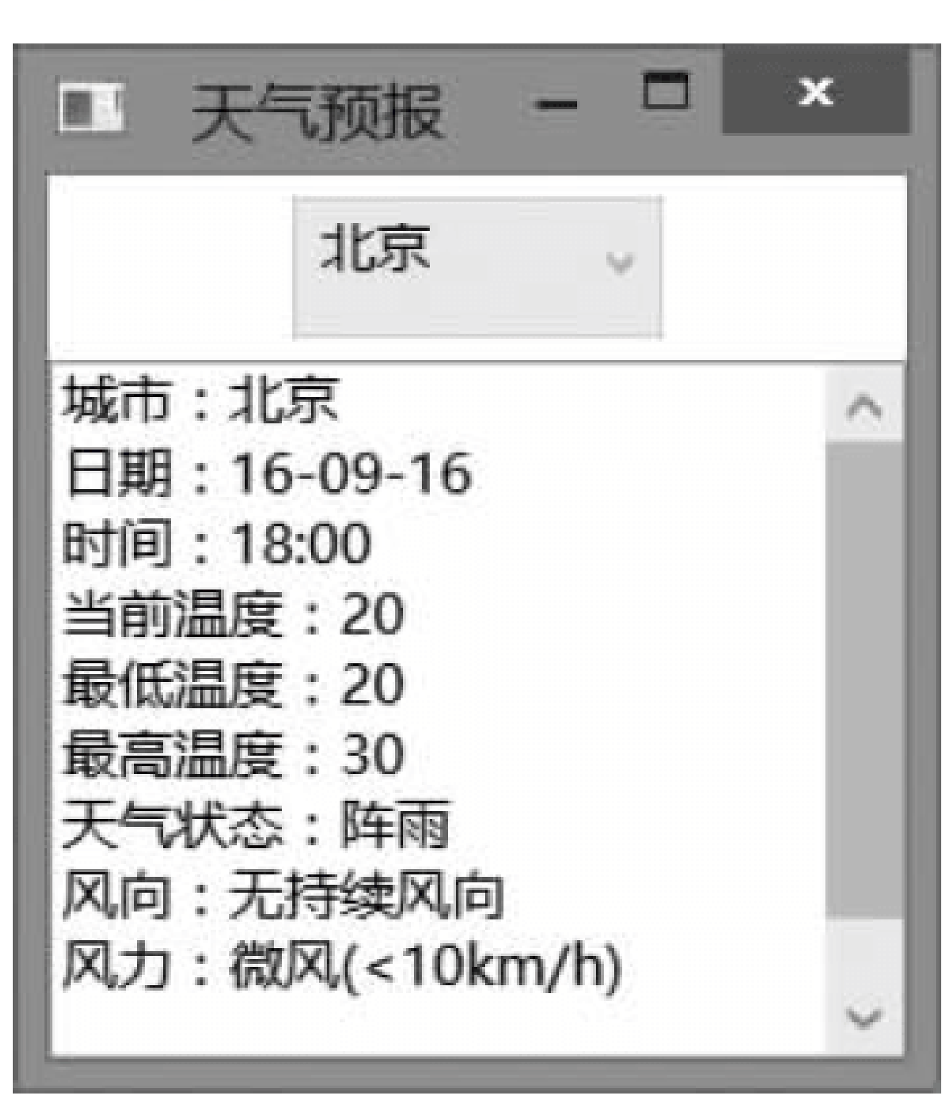


图 1-15　城市天气状况

主要的程序部分如下:

```
private void MyWindow_Loaded(object sender, RoutedEventArgs e)
{
    cbCity.Items.Add("选择");
    cbCity.Items.Add("北京");
    cbCity.Items.Add("上海");
    cbCity.Items.Add("广州");
    cbCity.Items.Add("深圳");
    client=new WebClient();
    client.Encoding=Encoding.UTF8;
    client.Headers["apikey"]=Properties.Resources.apikey;
    client.DownloadStringCompleted+=client_DownloadStringCompleted;
    cbCity.SelectedIndex=0;
}
private void cbCity_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if(cbCity.SelectedIndex>=1)
    {
        cbCity.IsEnabled=false;
        try
        {
            txt.Text="";
            String city=cbCity.SelectedItem.ToString();
              client.DownloadStringAsync (new Uri ("http://apis.baidu.com/
              apistore/weatherservice/cityname?cityname =" + city, UriKind.
              Absolute));
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
}
```

1.5 城市天气预报查询

1.5.1 案例展示

前面看到,可以从百度网站中获取一个城市当天的天气数据,实际上从百度网站中还可以得到城市几天内的天气预报数据。现在来设计一个程序,它启动后可以选择一个城市,显示这个城市几天内的天气预报数据,如图 1-16 所示。



图 1-16　城市几天内的天气预报

1.5.2　技术要点

1. 获取天气数据

根据百度天气预报的说明，可以访问下列网址获取一个城市多天的天气预报：

http://apis.baidu.com/apistore/weatherservice/recentweathers

要查询某个城市的天气预报，只要在网址的后面加上"cityname＝城市"的参数就可以了。

2. 解析天气数据

根据百度天气预报的说明，返回的是 JSON 数据。例如，北京的天气预报数据如下：

```
{
    errNum: 0,
    errMsg: "success",
    retData: {
        city: "北京",                          //城市名称
        cityid: "101010100",                   //城市编码
        today: {
            date: "2015-08-03",                //今天日期
            week: "星期一",                     //今日星期
            curTemp: "28℃",                    //当前温度
            aqi: "92",                         //空气质量指数
            fengxiang: "无持续风向",             //风向
            fengli: "微风级",                   //风力
            hightemp: "30℃",                   //最高温度
            lowtemp: "23℃",                    //最低温度
```

```
type: "阵雨",                    //天气状态
index: [                         //指标列表
    {
        name: "感冒指数",         //指数指标 1 名称
        code: "gm",              //指标编码
        index: "",               //等级
        details: "各项气象条件适宜,发生感冒机率较低。但请避免长期处于
        空调房间中,以防感冒。",     //描述
        otherName: ""            //其他信息
    },
    {
        code: "fs",
        details: "属中等强度紫外辐射天气,外出时应注意防护,建议涂擦 SPF
        指数高于 15,PA+的防晒护肤品。",
        index: "中等",
        name: "防晒指数",
        otherName: ""
    },
    {
        code: "ct",
        details: "天气炎热,建议着短衫、短裙、短裤、薄型 T 恤衫等清凉夏季
        服装。",
        index: "炎热",
        name: "穿衣指数",
        otherName: ""
    },
    {
        code: "yd",
        details: "有降水,推荐您在室内进行低强度运动;若坚持户外运动,须
        注意选择避雨防滑并携带雨具。",
        index: "较不宜",
        name: "运动指数",
        otherName: ""
    },
    {
        code: "xc",
        details: "不宜洗车,未来 24 小时内有雨,如果在此期间洗车,雨水和路
        上的泥水可能会再次弄脏您的爱车。",
        index: "不宜",
        name: "洗车指数",
        otherName: ""
    },
    {
        code: "ls",
```

```
            details: "有降水,不适宜晾晒。若需要晾晒,请在室内准备出充足的
            空间。",
            index: "不宜",
            name: "晾晒指数",
            otherName: ""
        }
    ]
},
forecast: [                              //回来预报列表
    {
        date: "2015-08-04",
        week: "星期二",
        fengxiang: "无持续风向",
        fengli: "微风级",
        hightemp: "32℃",
        lowtemp: "23℃",
        type: "多云"
    },
    {
        date: "2015-08-05",
        week: "星期三",
        fengxiang: "无持续风向",
        fengli: "微风级",
        hightemp: "30℃",
        lowtemp: "23℃",
        type: "多云"
    },
    {
        date: "2015-08-06",
        week: "星期四",
        fengxiang: "无持续风向",
        fengli: "微风级",
        hightemp: "29℃",
        lowtemp: "24℃",
        type: "雷阵雨"
    },
    {
        date: "2015-08-07",
        week: "星期五",
        fengxiang: "无持续风向",
        fengli: "微风级",
        hightemp: "30℃",
        lowtemp: "24℃",
        type: "多云"
```

```
            }
        ],
        history: [                          //历史天气列表

            {
                date: "2015-07-31",
                week: "星期五",
                aqi: "52",
                fengxiang: "无持续风向",
                fengli: "微风级",
                hightemp: "高温 29℃",
                lowtemp: "低温 22℃",
                type: "多云"
            },
            {
                date: "2015-08-01",
                week: "星期六",
                aqi: null,
                fengxiang: "南风",
                fengli: "微风级",
                hightemp: "高温 35℃",
                lowtemp: "低温 26℃",
                type: "多云"
            }
        ]
    }
}
```

观察这个 JSON 字符串，可以看到，forecas 是一个数组，它内部的每日天气数据是基本的数据，可以设计为一个 WeatherDataClass 类。retData 中包含当天的数据及未来几天的数据，因此可以定义 retData 是一个 WeatherCityClass 类，这个类包含一个 WeatherDataClass 对象 today 以及一个 WeatherDataClass 数组 forecast。最后定义 WeacherClass 类作为最外层的数据类，它包含返回信息与 WeatherCityClass 对象 retData。

我们不一定关心所有的天气数据，根据自己需要的信息可以设计一个类与该数据对应，这些类存储在文件 WeatherClass. cs 中，主要的类如下：

```
public class WeatherDataClass
{
    public String date { get; set; }
    public String week { get; set; }
    public String curTemp { get; set; }
    public String aqi { get; set; }
    public String fengxiang { get; set; }
    public String fengli { get; set; }
```

```
        public String hightemp { get; set; }
        public String lowtemp { get; set; }
        public String type { get; set; }
        public override string ToString()
        {
            String s="日期: "+date+" "+week+"\r\n";
            s=s+"天气状态: "+type+"\r\n";
            s=s+"风向: "+fengxiang+"\r\n 风力: "+fengli+"\r\n";
            s=s+"最低温度: "+lowtemp+"\r\n 最高温度: "+hightemp+"\r\n";
            return s;
        }
    }
    public class WeatherCityClass
    {
        public String city { get; set; }
        public String cityid { get; set; }
        public WeatherDataClass today { get; set; }
        public List<WeatherDataClass> forecast { get; set; }
    }
    public class WeatherClass
    {
        public int errNum { get; set; }
        public String errMsg { get; set; }
        public WeatherCityClass retData { get; set; }
    }
```

其中 WeatherDataClass 类是天气数据类，对于当天的天气数据有 curTemp 与 aqi 两个属性的值，对于未来几天的天气，这两个值为空。

1.5.3 服务器程序

服务器程序是百度天气预报网站 http://apis.baidu.com/apistore/weatherservice/recentweathers。

1.5.4 客户端程序

1. 界面设计

窗体中有一个名为 cbCity 的 ComboBox 下拉列表框与一个名称为 txt 的 TextBox 文本框。

```
<Window x:Name="MainWindow1" x:Class="client.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="天气预报" Height="350" Width="525" Loaded="MainWindow_
```

```
        Loaded">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="50" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <ComboBox x:Name="cbCity"  Width="60" Height="30" SelectionChanged
        ="cbCity_SelectionChanged"  />
        <TextBox x:Name="txt" Grid.Row="1" AcceptsReturn="True" IsReadOnly=
        "True" VerticalScrollBarVisibility="Auto" />
    </Grid>
</Window>
```

2. 程序设计

服务器程序在启动时执行窗体的 Loaded 函数，在这个函数中设置下拉列表框中包括的城市，同时创建 WebClient 对象，在选择城市时执行下拉列表框的 SelectionChanged 事件，选择城市就可以访问服务器获取天气预报数据。

```
public partial class MainWindow : Window
{
    WebClient client;
    public MainWindow()
    {
        InitializeComponent();
    }
    T deserialize<T>(String s)
    {
        DataContractJsonSerializer json=new DataContractJsonSerializer(typeof(T));
        byte[] buf=Encoding.UTF8.GetBytes(s);
        MemoryStream ms=new MemoryStream(buf);
        T obj=(T)json.ReadObject(ms);
        return obj;
    }
    void showMsg(String s)
    {
        txt.AppendText(s+"\r\n");
    }
    void client_DownloadStringCompleted(object sender,
    DownloadStringCompletedEventArgs e)
    {
        try
        {
            WeatherClass w=deserialize<WeatherClass>(e.Result);
```

```
            if(w.errNum==0)
            {
                String s="城市："+w.retData.city+"\r\n";
                s=s+w.retData.today.ToString()+"\r\n";
                foreach (WeatherDataClass wf in w.retData.forecast)
                    s=s+wf.ToString()+"\r\n";
                showMsg(s);
            }
            else showMsg(w.errMsg);
        }
        catch(Exception exp) { showMsg(exp.Message); }
        cbCity.IsEnabled=true;
    }
    private void cbCity_SelectionChanged(object sender,
    SelectionChangedEventArgs e)
    {
        if(cbCity.SelectedIndex>=1)
        {
            cbCity.IsEnabled=false;
            try
            {
                txt.Text="";
                String city=cbCity.SelectedItem.ToString();
                client.DownloadStringAsync(new Uri ("http://apis.baidu.com/
                apistore/weatherservice/recentweathers? cityname =" + city,
                UriKind.Absolute));
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
    private void MainWindow_Loaded(object sender, RoutedEventArgs e)
    {
        cbCity.Items.Add("选择");
        cbCity.Items.Add("北京");
        cbCity.Items.Add("上海");
        cbCity.Items.Add("广州");
        cbCity.Items.Add("深圳");
        client=new WebClient();
        client.Encoding=Encoding.UTF8;
        client.Headers["apikey"]=Properties.Resources.apikey;
        client.DownloadStringCompleted+=client_DownloadStringCompleted;
        cbCity.SelectedIndex=0;
    }
}
```

1.5.5 拓展训练

重新设计 WeatherCityClass 类，在其中增加一个 history 属性：

```
List<WeatherDataClass>histoty {get; set; }
```

那么它就能获取城市的历史天气数据，即 WeatherCityClass 如下：

```
public class WeatherCityClass
{
    public String city { get; set; }
    public String cityid { get; set; }
    public WeatherDataClass today { get; set; }
    public List<WeatherDataClass>forecast { get; set; }
    public List<WeatherDataClass>history { get; set; }
}
```

适当修改程序，就可以获取一个城市过去几天与未来几天的天气数据。

1.6 城市天气预报程序的实现

1.6.1 技术要点

1. 天气图标

前面的天气预报数据是文本的数据，为了让天气预报程序美观些，需要配上一些天气状况的图标，例如雨天的图标、晴天的图标等。天气状况的图标文件可以通过网络搜索下载，例如从网址 http://www.webxml.com.cn/zh_cn/weather_icon.aspx 就可以下载一组天气状况图标。这些图标如表 1-1 所示，每个图标有一个文件名称。

表 1-1 天气图标

天气状况	图标文件	图标样式
晴	a_0.gif	
多云	a_1.gif	
阴	a_2.gif	
阵雨	a_3.gif	
雷阵雨	a_4.gif	
雷阵雨-冰雹	a_5.gif	
雨夹雪	a_6.gif	

续表

天气状况	图标文件	图标样式
小雨	a_7.gif	
中雨	a_8.gif	
大雨	a_9.gif	
暴雨	a_10.gif	
大暴雨	a_11.gif	
特大暴雨	a_12.gif	
阵雪	a_13.gif	
小雪	a_14.gif	
中雪	a_15.gif	
大雪	a_16.gif	
暴雪	a_17.gif	
雾	a_18.gif	
冻雨	a_19.gif	
沙尘暴	a_20.gif	
小雨-中雨	a_21.gif	
中雨-大雨	a_22.gif	
大雨-暴雨	a_23.gif	
暴雨-大暴雨	a_24.gif	
大暴雨-特大暴雨	a_25.gif	
小雪-中雪	a_26.gif	
中雪-大雪	a_27.gif	
大雪-暴雪	a_28.gif	
浮尘	a_29.gif	
扬沙	a_30.gif	
强沙尘暴	a_31.gif	

可以为每个天气状况与对应的图标建立一个字典关系，这样在知道天气状况时就可以查找到对应的图标是什么，例如：

```
icons=new Dictionary<string, string>();
icons.Add("晴", "a_0.gif");
icons.Add("多云", "a_1.gif");
icons.Add("阴", "a_2.gif");
```

所有的图标可以存储在工程项目的 icons 目录下，如图 1-17 所示，只要在 Weathers 项目中新建一个 icons 文件夹，把这些天气图标加入到这个文件夹即可。



图 1-17 天气图标

2. 天气数据类

数据类与 1.5 节的类似，只是在 WeatherDataClass 中增加了一个 icon 的属性，用来显示天气图标。

```
public class WeatherDataClass
{
    public String date { get; set; }
    public String week { get; set; }
    public String curTemp { get; set; }
    public String aqi { get; set; }
    public String fengxiang { get; set; }
    public String fengli { get; set; }
    public String hightemp { get; set; }
    public String lowtemp { get; set; }
    public String type { get; set; }
    public String icon { get; set; }
}
public class WeatherCityClass
{
    public String city { get; set; }
    public String cityid { get; set; }
    public WeatherDataClass today { get; set; }
    public List<WeatherDataClass>forecast { get; set; }
}
public class WeatherClass
{
    public int errNum { get; set; }
    public String errMsg { get; set; }
    public WeatherCityClass retData { get; set; }
}
```

为了查询全国各个省份与城市的天气，另外设计 ProvinceClass 来表示一个省份，它下面有一个城市列表 cities，它是这个省所辖的各个城市。设计 ZoneClass 类来表示全部省份，它有一个省份的列表 provinces。

```
public class ProvinceClass
{
    public String province { get; set;  }
    public List<String>cities { get; set; }
}
public class ZoneClass
```

```
{
    public List<ProvinceClass>provinces { get; set; }
}
```

1.6.2　服务器程序

服务器程序是百度天气预报网站 http://apis.baidu.com/apistore/weatherservice/recentweathers。

1.6.3　客户端程序

1. 界面设计

程序把窗体划分成 5 个主要的列，第一列存放省份与城市的列表，其他 4 个部分存放 4 天的天气预报数据，每个天气面板是一个 StackPanel 控件，其中 firstWeather、secondWeather、thirdWeather、fourthWeather 分别是第一、二、三、四天的数据数据面板，每天的数据都绑定在这个面板的 DataContext 属性上。

```
<Window x:Name="MainWindow1" x:Class="Weathers.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="天气预报" Height="322.56" Width="676.139" Loaded="MainWindow_
        Loaded" ResizeMode="NoResize">
    <Grid>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="40" />
                <RowDefinition Height="20" />
                <RowDefinition Height="*" />
            </Grid.RowDefinitions>
            <TextBlock Text="天气预报" FontSize="30" HorizontalAlignment=
            "Center" />
            <TextBlock x:Name="tbZone" Text="" FontSize="14"
            HorizontalAlignment="Center" Grid.Row="1" />
            <Grid Grid.Row="2">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="100" />
                    <ColumnDefinition Width="140" />
                    <ColumnDefinition Width="140" />
                    <ColumnDefinition Width="140" />
                    <ColumnDefinition Width="140" />
                </Grid.ColumnDefinitions>
                <Grid>
                    <Grid.RowDefinitions>
```

```
            <RowDefinition Height="30" />
            <RowDefinition Height="30" />
            <RowDefinition Height="30" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <TextBlock Text="省份" Grid.Row="0" Margin="10,0,0,0" />
        <ComboBox x:Name="cbProvince"  Grid.Row="1" Margin="10,0,
        0,0" SelectionChanged="cbProvince_SelectionChanged" />
        <TextBlock Text="城市"  Grid.Row="2" Margin="10,0,0,0" />
        <ListBox x:Name="lsCities"  Grid.Row="3" Margin="10,0,0,
        10" SelectionChanged="lsCities_SelectionChanged" />
    </Grid>
    <StackPanel x:Name="firstWeather" Orientation="Vertical"
    Grid.Column="1" Margin="10">
        <TextBlock FontSize="10" Text="{Binding date}" />
        <TextBlock FontSize="10" Text="{Binding week}" />
        <TextBlock FontSize="10" Text="{Binding curTemp}" />
        <TextBlock FontSize="10" Text="{Binding aqi}" />
        <TextBlock FontSize="10" Text="{Binding lowtemp}" />
        <TextBlock FontSize="10" Text="{Binding hightemp}" />
        <TextBlock FontSize="10" Text="{Binding fengxiang}" />
        <TextBlock FontSize="10" Text="{Binding fengli}" />
        <TextBlock FontSize="10" Text="{Binding type}" />
        <Image Width="80" Height="80" Source="{Binding icon}" />
    </StackPanel>
    <StackPanel x:Name="secondWeather" Orientation="Vertical"
    Grid.Column="2" Margin="10">
        <TextBlock FontSize="10" Text="{Binding date}" />
        <TextBlock FontSize="10" Text="{Binding week}" />
        <TextBlock FontSize="10" Text="{Binding curTemp}" />
        <TextBlock FontSize="10" Text="{Binding aqi}" />
        <TextBlock FontSize="10" Text="{Binding lowtemp}" />
        <TextBlock FontSize="10" Text="{Binding hightemp}" />
        <TextBlock FontSize="10" Text="{Binding fengxiang}" />
        <TextBlock FontSize="10" Text="{Binding fengli}" />
        <TextBlock FontSize="10" Text="{Binding type}" />
        <Image Width="80" Height="80" Source="{Binding icon}" />
    </StackPanel>
    <StackPanel x:Name="thirdWeather" Orientation="Vertical"
    Grid.Column="3" Margin="10">
        <TextBlock FontSize="10" Text="{Binding date}" />
        <TextBlock FontSize="10" Text="{Binding week}" />
        <TextBlock FontSize="10" Text="{Binding curTemp}" />
        <TextBlock FontSize="10" Text="{Binding aqi}" />
```

```
                <TextBlock FontSize="10" Text="{Binding lowtemp}" />
                <TextBlock FontSize="10" Text="{Binding hightemp}" />
                <TextBlock FontSize="10" Text="{Binding fengxiang}" />
                <TextBlock FontSize="10" Text="{Binding fengli}" />
                <TextBlock FontSize="10" Text="{Binding type}" />
                <Image Width="80" Height="80" Source="{Binding icon}" />
            </StackPanel>
            <StackPanel x:Name="fourthWeather" Orientation="Vertical"
            Grid.Column="4" Margin="10">
                <TextBlock FontSize="10" Text="{Binding date}" />
                <TextBlock FontSize="10" Text="{Binding week}" />
                <TextBlock FontSize="10" Text="{Binding curTemp}" />
                <TextBlock FontSize="10" Text="{Binding aqi}" />
                <TextBlock FontSize="10" Text="{Binding lowtemp}" />
                <TextBlock FontSize="10" Text="{Binding hightemp}" />
                <TextBlock FontSize="10" Text="{Binding fengxiang}" />
                <TextBlock FontSize="10" Text="{Binding fengli}" />
                <TextBlock FontSize="10" Text="{Binding type}" />
                <Image Width="80" Height="80" Source="{Binding icon}" />
            </StackPanel>
        </Grid>
    </Grid>
  </Grid>
</Window>
```

2. 程序设计

程序启动时，首先建立天气图标字典 icons、省份与城市的对象 zones，然后把省份列表绑定到省份下拉列表框 cbProvince 上显示不同的省份。当省份变化时，获取该省份所辖的城市列表并绑定到 lsCities 列表框中显示该省份的城市。当选择一个城市时，就从服务器获取天气预报数据，分别显示在不同的面板上。其中 getWeatherItem 函数是给 WeatherDataClass 对象的各个天气数据加上中文说明，它们绑定到任何一个天气面板上就有说明信息了。

```
public partial class MainWindow : Window
    {
        WebClient client;
        String url="http://apis.baidu.com/apistore/weatherservice/
        recentweathers?cityname=";
        Dictionary<String, String>icons;
        ZoneClass zones;
        public MainWindow()
        {
            InitializeComponent();
```

```
        client=new WebClient();
        client.Headers["apikey"]=Properties.Resources.apikey;
        client.DownloadStringCompleted+=client_DownloadStringCompleted;
    }
    void client_DownloadStringCompleted(object sender,
    DownloadStringCompletedEventArgs e)
    {
        if(e.Error==null)
        {
            try
            {
                tbZone.Text=e.UserState.ToString();
                String s=e.Result;
                //反序列化成Weather对象
                WeatherClass ws=deserialize<WeatherClass>(s);
                if(ws.errNum==0)
                {
                    //绑定数据到第一天的显示控件
                    WeatherDataClass w=ws.retData.today;
                    getWeatherItem(w);
                    firstWeather.DataContext=w;
                    if(ws.retData.forecast.Count>0)
                    {
                        //绑定数据到第二天的显示控件
                        w=ws.retData.forecast[0];
                        getWeatherItem(w);
                        secondWeather.DataContext=w;
                    }
                    if(ws.retData.forecast.Count>1)
                    {
                        //绑定数据到第三天的显示控件
                        w=ws.retData.forecast[1];
                        getWeatherItem(w);
                        thirdWeather.DataContext=w;
                    }
                    if(ws.retData.forecast.Count>2)
                    {
                        //绑定数据到第四天的显示控件
                        w=ws.retData.forecast[2];
                        getWeatherItem(w);
                        fourthWeather.DataContext=w;
                    }
                }
                else
```

```
                {
                    //绑定空数据到各天的显示控件
                    firstWeather.DataContext=null;
                    secondWeather.DataContext=null;
                    thirdWeather.DataContext=null;
                    fourthWeather.DataContext=null;
                    showMsg(ws.errMsg);
                }
            }
            catch(Exception exp)
            {
                MessageBox.Show(exp.Message);
            }
            cbProvince.IsEnabled=true;
            lsCities.IsEnabled=true;
        }
    }
    void showMsg(String s)
    {
        MessageBox.Show(s, "Information", MessageBoxButton.OK);
    }
    T deserialize<T>(String s)
    {
        byte[] buf=Encoding.UTF8.GetBytes(s);
        MemoryStream ms=new MemoryStream(buf);
        DataContractJsonSerializer json=new DataContractJsonSerializer
        (typeof(T));
        T obj=(T)json.ReadObject(ms);
        return obj;
    }
    void getWeatherItem(WeatherDataClass w)
    {
        //查询天气图标
        KeyValuePair<String, String>obj=icons.FirstOrDefault(x=>x.Key.
        IndexOf(w.type)>=0 || w.type.IndexOf(x.Key)>=0);
        if(!obj.Equals(null)) w.icon="icons\\"+obj.Value;
        else w.icon=null;
        //w.date       ="日期：      "+w.date;
        //w.week       ="星期：      "+w.week;
        w.curTemp      ="现在温度：  "+w.curTemp;
        w.aqi          ="空气指数：  "+w.aqi;
        w.hightemp     ="最高温度：  "+w.hightemp;
        w.lowtemp      ="最低温度：  "+w.lowtemp;
        w.fengxiang    ="风向：      "+w.fengxiang;
```

```
    w.fengli      ="风力:      "+w.fengli;
    w.type        ="天气状况: "+w.type;
}
void getIcons()
{
    icons=new Dictionary<string, string>();
    icons.Add("晴", "a_0.gif");
    icons.Add("多云", "a_1.gif");
    icons.Add("阴", "a_2.gif");
    icons.Add("阵雨", "a_3.gif");
    icons.Add("雷阵雨", "a_4.gif");
    icons.Add("雷阵雨-冰雹", "a_5.gif");
    icons.Add("雨加雪", "a_6.gif");
    icons.Add("小雨", "a_7.gif");
    icons.Add("中雨", "a_8.gif");
    icons.Add("大雨", "a_9.gif");
    icons.Add("暴雨", "a_10.gif");
    icons.Add("大暴雨", "a_11.gif");
    icons.Add("特大暴雨", "a_12.gif");
    icons.Add("阵雪", "a_13.gif");
    icons.Add("小雪", "a_14.gif");
    icons.Add("中雪", "a_15.gif");
    icons.Add("大雪", "a_16.gif");
    icons.Add("暴雪", "a_17.gif");
    icons.Add("雾", "a_18.gif");
    icons.Add("冻雨", "a_19.gif");
    icons.Add("沙尘暴", "a_20.gif");
    icons.Add("小雨-中雨", "a_21.gif");
    icons.Add("中雨-大雨", "a_22.gif");
    icons.Add("大雨-暴雨", "a_23.gif");
    icons.Add("暴雨-大暴雨", "a_24.gif");
    icons.Add("大暴雨-特大暴雨", "a_25.gif");
    icons.Add("小雪-中雪", "a_26.gif");
    icons.Add("中雪-大雪", "a_27.gif");
    icons.Add("大雪-暴雪", "a_28.gif");
    icons.Add("浮尘", "a_29.gif");
    icons.Add("扬沙", "a_30.gif");
    icons.Add("强沙尘暴", "a_31.gif");
}
void getZones()
{
    zones=new ZoneClass
    {
        provinces=new List<ProvinceClass>
```

```
            {
                new ProvinceClass { province="直辖市", cities=new List
                    <string>{ "北京", "上海", "天津", "重庆" } },
                new ProvinceClass { province="广东", cities=new List
                    <string>{ "广州", "深圳", "珠海", "惠州" } },
                new ProvinceClass { province="贵州", cities=new List
                    <string>{ "贵阳", "六枝", "兴义", "遵义" } }
            }
        };
    }
    private void MainWindow_Loaded(object sender, RoutedEventArgs e)
    {
        try
        {
            getIcons();
            getZones();
            //绑定省份列表到 cbProvince 控件
            cbProvince.ItemsSource=zones.provinces.Select(x=>x.province);
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
    private void lsCities_SelectionChanged(object sender, Selection-
    ChangedEventArgs e)
    {
        if(lsCities.SelectedIndex>=0)
        {
            String city=lsCities.SelectedItem.ToString();
            //查找 city 的天气预报
            client.DownloadStringAsync(new Uri(url+city, UriKind.Absolute),
            cbProvince.SelectedItem.ToString()+" "+city);
            cbProvince.IsEnabled=false;
            lsCities.IsEnabled=false;
        }
    }
    private void cbProvince_SelectionChanged(object sender,
    SelectionChangedEventArgs e)
    {
        //省份变化时重新绑定城市列表到 lsCities 控件
        if(cbProvince.SelectedIndex>=0)
            lsCities.ItemsSource=zones.provinces[cbProvince.
            SelectedIndex].cities;
        else lsCities.ItemsSource=null;
    }
}
```

值得注意的是查找天气图标的语句：

```
KeyValuePair<String, String>obj=icons.FirstOrDefault(x=>x.Key.
IndexOf(w.type)>=0 || w.type.IndexOf(x.Key)>=0);
```

其中 x. Key. IndexOf(w. type)是在 icons 中查找包含 w. type 字样的一个项，而 w. type. IndexOf(x. Key)是在 w. type 中查找是否包含 icons 中的一个项，它们中的任何一项成立时就确定一个天气图标。

1.6.4 拓展训练

这个天气预报程序是一个 WPF 窗体程序，实际上也可以类似地做一个网页版的天气预报程序，以 Web 目录为网站，在 Web 下建立 icons 目录，把图标文件放在 icons 中，在 Web 中添加一个网页文件 Default. aspx。

1. 界面设计

Default. aspx 是界面程序，界面包括一个省份下拉列表框与城市下拉列表框，同时包含一个可以实现数据重复显示的 DataList 控件，用来显示多天的天气预报数据。Default. aspx 界面中主要包含的控件功能如表 1-2 所示。

表 1-2 界面控件功能

控件类型	控件 ID	功 能 说 明
DropDownList	cbProvince	绑定省份的列表，当省份发生变化时触发 SelectionChanged 事件，获取该省份下辖的所有城市，绑定城市到 cbCity 控件
DropDownList	cbCity	绑定城市的列表，当城市发生变化时触发 SelectionChanged 事件，获取该城市的天气预报数据，把天气预报数据绑定到 list 控件
DataList	list	这个控件包含一个<ItemTemplate>的数据模板，其中包含一天的天气预报数据，用 DataList 实现模板的重复显示，从而显示出几天的天气预报数据

程序采用网络异步操作，因此要设置 Async 为 true，Default. aspx 界面程序如下：

```
<%@ Page Language="C# " AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" Async="true" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="myform" runat="server">
    <div>
```

```
    省份<asp:DropDownList ID="cbProvince" runat="server" AutoPostBack="true"
            onselectedindexchanged="cbProvince_SelectedIndexChanged"
            ondatabound="cbProvince_DataBound"></asp:DropDownList>
    城市<asp:DropDownList ID="cbCity" runat="server" AutoPostBack="true"
            onselectedindexchanged="cbCity_SelectedIndexChanged"
            ondatabound="cbCity_DataBound"></asp:DropDownList>
    选择城市<asp:Label ID="selectedCity" runat="server" />
    </div>
    <p></p>
    <div>
    <asp:DataList ID="list" runat="server" RepeatColumns="5"
    RepeatDirection="Horizontal" RepeatLayout="Table" Font-Size="12px">
      <ItemTemplate>
      <asp:panel Width="168" runat="server">
        <asp:Label ID="date" runat="server" Text='<%# Eval("date") %>' /><br />
        <asp:Label ID="week" runat="server" Text='<%# Eval("week")%>' /><br />
        <asp:Label ID="curtemp" runat="server" Text='<%# Eval("curtemp") %>' />
        <br />
        <asp:Label ID="aqi" runat="server" Text='<%# Eval("aqi")%>' /><br />
        <asp:Label ID="lowtemp" runat="server" Text='<%# Eval("lowtemp") %>' />
        <br />
        <asp:Label ID="hightemp" runat="server" Text='<%# Eval("hightemp")%>' />
        <br />
        <asp:Label ID="fengxiang" runat="server" Text='<%# Eval("fengxiang")
        %>' /><br />
        <asp:Label ID="fengli" runat="server" Text='<%# Eval("fengli")%>' />
        <br />
        <asp:Label ID="type" runat="server" Text='<%# Eval("type") %>' /><br />
        <asp:Image ID="icon" runat="server" Width="50" Height="50" ImageUrl
        ='<%# Eval("icon")%>' /><br />
      </asp:panel>
    </ItemTemplate>
    </asp:DataList>
    </div>
    <asp:Label ID="msg" runat="server" />
    </form>
</body>
</html>
```

2. 程序设计

程序在启动时先调用 getZones 函数获取省份与城市的数据列表 zones，调用 getIcon 函数获取图标字典 icons，然后把省份列表绑定到 cbProvince，然后执行 cbProvince_

DataBound 函数，获取省份并获取城市，绑定城市列表到 cbCity，然后执行 cbCity_DataBound 函数，获取城市，然后调用 getForecast 获取天气预报数据。注意在 getForecast 中异步调用网络下载函数 DownloadStringAsync，当异步调用完成后执行 DownloadStringCompleted 函数，在此函数中解析天气预报数据，获取多天的天气预报数据列表，绑定到 DataList 控件上，完成天气预报数据显示。Default.aspx.cs 文件如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Net;
using System.Text;
using System.IO;
using System.Runtime.Serialization.Json;
public class WeatherDataClass
{
    public String curTemp { get; set; }
    public String aqi { get; set; }
    public String date { get; set; }
    public String week { get; set; }
    public String fengxiang { get; set; }
    public String fengli { get; set; }
    public String hightemp { get; set; }
    public String lowtemp { get; set; }
    public String type { get; set; }
    public String icon { get; set; }
}
public class WeatherCityClass
{
    public String city { get; set; }
    public String cityid { get; set; }
    public WeatherDataClass today { get; set; }
    public List<WeatherDataClass> forecast { get; set; }
}
public class WeatherClass
{
    public int errNum { get; set; }
    public String errMsg { get; set; }
    public WeatherCityClass retData { get; set; }
}
public class ProvinceClass
{
```

```
    public String province { get; set; }
    public List<String>cities { get; set; }
}
public class ZoneClass
{
    public List<ProvinceClass>provinces { get; set; }
}
public partial class _Default : System.Web.UI.Page
{
    WebClient client;
    String url="http://apis.baidu.com/apistore/weatherservice/recentweathers?
    cityname=";
    Dictionary<String, String>icons;
    ZoneClass zones;
    void getWeatherItem(WeatherDataClass w)
    {
        //查询天气图标
        if(w.type !=null)
        {
            KeyValuePair<String, String>obj=icons.FirstOrDefault(x=>x.Key.
            IndexOf(w.type)>=0 || w.type.IndexOf(x.Key)>=0);
            if(!obj.Equals(null)) w.icon="icons\\"+obj.Value;
        }
        else w.icon =null;
        w.curTemp   ="现在温度:  "+w.curTemp;
        w.aqi       ="空气指数:  "+w.aqi;
        w.date      ="日期:      "+w.date;
        w.week      ="星期:      "+w.week;
        w.hightemp  ="最高温度:  "+w.hightemp;
        w.lowtemp   ="最低温度:  "+w.lowtemp;
        w.fengxiang="风向:       "+w.fengxiang;
        w.fengli    ="风力:      "+w.fengli;
        w.type      ="天气状况:  "+w.type;
    }
    void getIcons()
    {
        icons=new Dictionary<string, string>();
        icons.Add("晴", "a_0.gif");
        icons.Add("多云", "a_1.gif");
        icons.Add("阴", "a_2.gif");
        icons.Add("阵雨", "a_3.gif");
        icons.Add("雷阵雨", "a_4.gif");
        icons.Add("雷阵雨-冰雹", "a_5.gif");
        icons.Add("雨夹雪", "a_6.gif");
```

```
        icons.Add("小雨", "a_7.gif");
        icons.Add("中雨", "a_8.gif");
        icons.Add("大雨", "a_9.gif");
        icons.Add("暴雨", "a_10.gif");
        icons.Add("大暴雨", "a_11.gif");
        icons.Add("特大暴雨", "a_12.gif");
        icons.Add("阵雪", "a_13.gif");
        icons.Add("小雪", "a_14.gif");
        icons.Add("中雪", "a_15.gif");
        icons.Add("大雪", "a_16.gif");
        icons.Add("暴雪", "a_17.gif");
        icons.Add("雾", "a_18.gif");
        icons.Add("冻雨", "a_19.gif");
        icons.Add("沙尘暴", "a_20.gif");
        icons.Add("小雨-中雨", "a_21.gif");
        icons.Add("中雨-大雨", "a_22.gif");
        icons.Add("大雨-暴雨", "a_23.gif");
        icons.Add("暴雨-大暴雨", "a_24.gif");
        icons.Add("大暴雨-特大暴雨", "a_25.gif");
        icons.Add("小雪-中雪", "a_26.gif");
        icons.Add("中雪-大雪", "a_27.gif");
        icons.Add("大雪-暴雪", "a_28.gif");
        icons.Add("浮尘", "a_29.gif");
        icons.Add("扬沙", "a_30.gif");
        icons.Add("强沙尘暴", "a_31.gif");
    }
    void getZones()
    {
        zones=new ZoneClass
        {
            provinces=new List<ProvinceClass>
                {
                    new ProvinceClass { province="直辖市", cities=new List
                    <string>{ "北京", "上海", "天津", "重庆" } } ,
                    new ProvinceClass { province="广东", cities=new List
                    <string>{ "广州", "深圳", "珠海", "惠州" } } ,
                    new ProvinceClass { province="贵州", cities=new List
                    <string>{ "贵阳", "六枝", "兴义", "遵义" } }
                }
        };
    }
    T deserialize<T>(String s)
    {
        byte[] buf=Encoding.UTF8.GetBytes(s);
```

```
        MemoryStream ms=new MemoryStream(buf);
        DataContractJsonSerializer json=new DataContractJsonSerializer
        (typeof(T));
        T obj=(T)json.ReadObject(ms);
        return obj;
    }
    protected void Page_Load(object sender, EventArgs e)
    {
        msg.Text="";
        selectedCity.Text="";
        getIcons();
        getZones();
        client=new WebClient();
        client.DownloadStringCompleted+=new
        DownloadStringCompletedEventHandler(client_DownloadStringCompleted);
        client.Encoding=Encoding.UTF8;
        client.Headers["apikey"]="……";
        if(!Page.IsPostBack)
        {
            //绑定省份列表到 cbProvince 控件
            cbProvince.DataSource=zones.provinces.Select(x=>x.province);
            cbProvince.DataBind();
        }

    }
    void client_DownloadStringCompleted(object sender,
    DownloadStringCompletedEventArgs e)
    {
        if(e.Error==null)
        {
            WeatherClass ws=deserialize<WeatherClass>(e.Result);
            if(ws.errNum==0)
            {
                List<WeatherDataClass>wList=new List<WeatherDataClass>();
                //绑定数据到第一天的显示控件
                wList.Add(ws.retData.today);
                for (int i=0; i<ws.retData.forecast.Count; i++)
                    wList.Add(ws.retData.forecast[i]);
                for (int i=0; i<wList.Count; i++) getWeatherItem(wList[i]);
                list.DataSource=wList;
                list.DataBind();
                selectedCity.Text=e.UserState.ToString();
            }
```

```
            else msg.Text=ws.errMsg;
        }
        else msg.Text=e.Error.Message;
    }
    protected void cbProvince_SelectedIndexChanged(object sender, EventArgs e)
    {
        //省份变化时重新绑定城市列表到 lsCities 控件
        if(cbProvince.SelectedIndex>=0)
            cbCity.DataSource=zones.provinces[cbProvince.SelectedIndex].
            cities;
        else cbCity.DataSource=null;
        cbCity.DataBind();
    }
    protected void cbCity_SelectedIndexChanged(object sender, EventArgs e)
    {
        getForecast();
    }
    void getForecast()
    {
        if(cbCity.SelectedIndex>=0)
        {
            String city=cbCity.SelectedItem.ToString();
            //查找 city 的天气预报
            try
            {
                client.DownloadStringAsync(new Uri(url+city, UriKind.
                Absolute),city);
            }
            catch(Exception exp) { msg.Text=exp.Message; }
        }
    }
    protected void cbProvince_DataBound(object sender, EventArgs e)
    {
        cbProvince_SelectedIndexChanged(sender, e);
    }
    protected void cbCity_DataBound(object sender, EventArgs e)
    {
        getForecast();
    }
}
```

其中，在 client.Headers["apikey"] = "……"中要使用自己的 apikey 值。

练 习 一

1. 建立一个学生类 StudentClass，包括学号 sNo 及姓名 sName，完成下列操作：

(1) 创建一个学号为 1001，姓名为“张三”的学生对象，分别写出这个对象 XML 系列化与 JSON 序列化的字符串。

(2) 创建包含 3 个学生的对象列表，它们的学号和姓名分别是("1001","张三")，("1002","李四")，("1003","王五")，分别写出这个列表 XML 序列化与 JSON 序列化的字符串。

2. 有一个 XML 字符串如下：

```
<?xml version="1.0"?>
<ZoneClass>
  <provinces>
    <ProvinceClass>
      <province>直辖市</province>
      <cities>
        <string>北京</string>
        <string>上海</string>
        <string>天津</string>
        <string>重庆</string>
      </cities>
    </ProvinceClass>
    <ProvinceClass>
      <province>广东</province>
      <cities>
        <string>广州</string>
        <string>深圳</string>
        <string>珠海</string>
      </cities>
    </ProvinceClass>
    <ProvinceClass>
      <province>贵州</province>
      <cities>
        <string>贵阳</string>
        <string>六枝</string>
        <string>兴义</string>
      </cities>
    </ProvinceClass>
  </provinces>
</ZoneClass>
```

试编写适当的 ProvinceClass 类与 ZoneClass 类，并编写反序列化函数把这个字符串

反序列化成对应类的对象。

3. 有一个JSON字符串如下：

```
[{"cities":["北京","上海","天津","重庆"],"province":"直辖市"},
{"cities":["广州","深圳","珠海","惠州"],"province":"广东"},
{"cities":["贵阳","六枝","兴义","遵义"],"province":"贵州"}]
```

试编写适当数据类，并编写反序列化函数把这个字符串反序列化成对应类的对象。

4. 已知一个学生类：

```
public class Student
{
    public Name { get; set; }
    public Sex { get; set; }
}
```

如果有一个学生列表对象 students 如下：

```
List<Student>students=new List<Student>
{
    new Student {Name="张一",Sex="男"},
    new Student {Name="张二",Sex="女"},
    new Student {Name="张三",Sex="男"},
    new Student {Name="张四",Sex="女"}
};
```

请编写程序获取所有的女生名单，并把名单序列化成一个JSON字符串。

5. 一个学生数据库的 students 表存储了学生学号(SNO)与姓名(SNAME)，例如：

学　号	姓　名
1001	张三
1002	李四
⋮	⋮

(1) 设计服务器以JSON的格式返回所有学生的学号与姓名：

```
{students: [{"sno":"1001","sname":"张三"},{"sno":"1002","sname":"李四"},
… ] }
```

设计客户程序是一个WPF窗体程序，解析出学号与姓名并用表格的形式显示。

(2) 设计服务器以XML的格式返回所有学生的学号与姓名，例如：

```
<students>
<student><sno>1001</sno><sname>张三</sname></studnet>
<student><sno>1002</sno><sname>李四</sname></studnet>
⋮
```

```
</students>
```

设计客户程序是一个 ASP. NET 网页程序，解析出学号与姓名并用表格的形式显示。

6. 百度 API Store 中还有很多日常生活中有用的接口服务，其中手机归属地查询就是一个。编写一个程序调用这个服务，输入一个手机号码，显示该手机是归属哪个省份哪个城市的。

项目二



基于 Web 的图片共享程序

网络图片共享程序主要由服务器程序与客户端程序组成，服务器程序是一个网页程序，其功能是管理数据库中的图片，完成与客户端的交互。客户端功能是用 WPF 窗体的形式浏览和管理服务器中的图片。程序结构如图 2-1 所示，客户端与服务器主要通过 XML 格式的数据进行交互。

客户端程序连接服务器后可以浏览到服务器中所有用户共享的图片，如图 2-2 所示。如果要管理图片，可右击列表控件，弹出一个菜单执行登录操作，如图 2-3 所示。用户登录后可以对自己的图片进行增加、删除、共享等操作，如图 2-4 所示。只有标上共享标志的图片才能被其他用户浏览，没有共享的图片是本用户私密的，其他用户无权浏览。用户登录后也只能管理自己的图片，无权管理别的用户的图片，如图 2-5 所示。

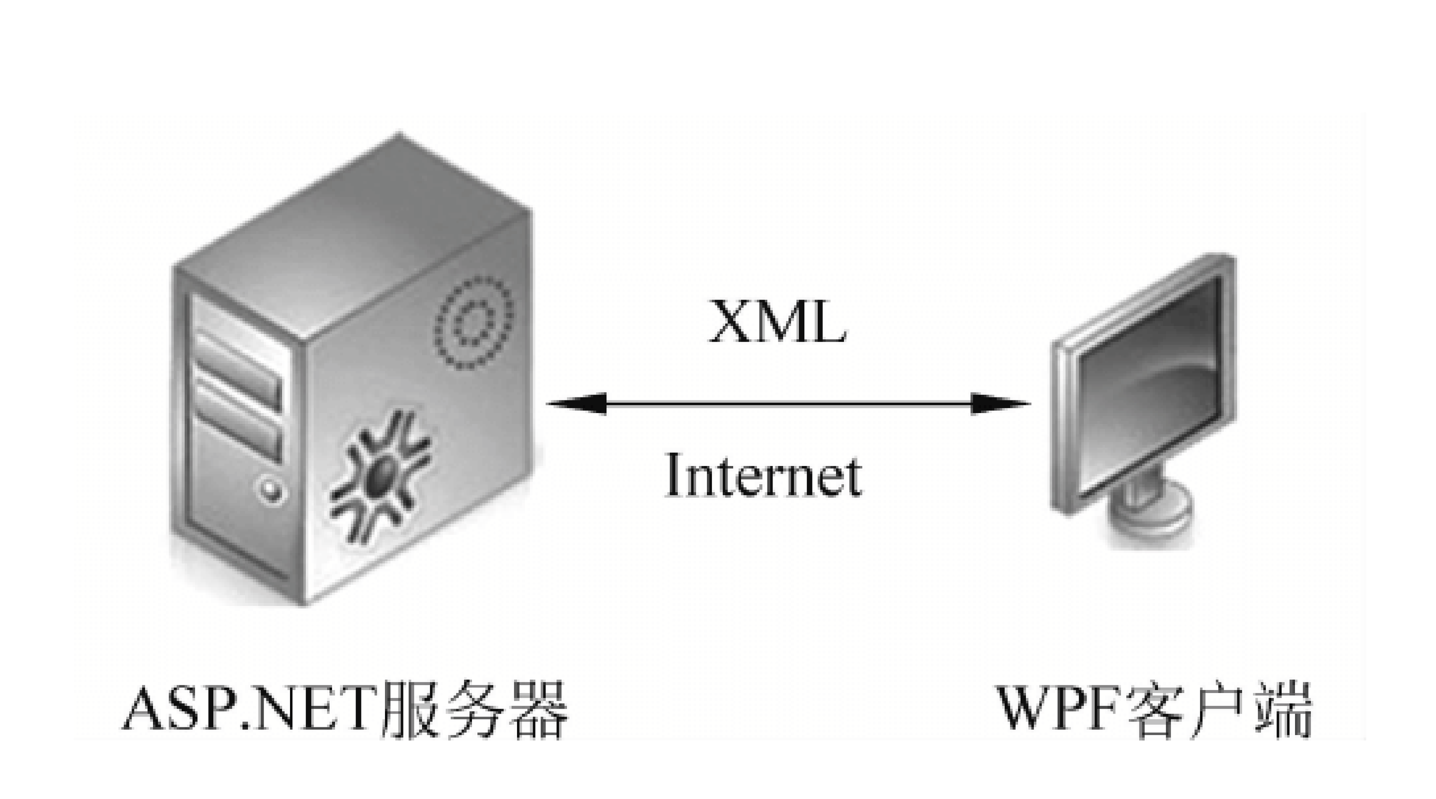


图 2-1　程序结构



图 2-2　客户端程序

用户登录
更改密码
上传图片
删除图片
设置共享
另存图片
刷新图片

图 2-3　弹出菜单

登录与注册
用户 xxx
密码 ●●●
取消 登录

图 2-4　登录窗体



图 2-5 设置图片共享

2.1 用户信息的发送

2.1.1 案例展示

设计服务器程序管理用户的登录或者注册,设计客户端完成用户信息的提交,服务器获取用户信息后返回给客服端,如图 2-6 所示。

2.1.2 技术要点

1. 用户信息



图 2-6 用户信息发送

客户端要向服务器传递用户的名称 uName 与密码 uPass 信息,这些信息可以放在地址后面传递,如果 url 是服务器的地址,那么地址格式为

```
url+"?uName="+uName+"&uPass="+uPass
```

参数通过地址传输给服务器,服务器通过 Response 对象获取 uName 与 uPass 的值,函数如下:

```
public void ProcessRequest(HttpContext context)
{
    String uName=context.Response.QueryString["uName"];
    String uPass=context.Response.QueryString["uPass"];
}
```

这个方法对于一般的用户数据都是可行的,但是也有例外,如果用户信息包含特殊符号,例如 uName="A&B"及 uPass="123",就会出现问题。因为地址后面的参数是用 & 分隔的,对应出现

```
url+"?uName=A&B&uPass=123"
```

这样的地址,服务器解析出来的 uName 值是"A"而不是"A&B",这样就出现错误了。

为了防止这种错误的发生，一般地址参数要用函数进行编码与解码，例如使用UrlEncode与UrlDecode函数编码与解码，也可以自己定义编码与解码函数。由于UrlEncode与UrlDecode函数是定义在System.Web空间的Server服务器对象中的，在客户端使用不方便，因此本节采用自己设计的编码与解码函数。

一种简单的编码方法是把字符串按UTF8编码转为二进制数据，然后把二进制数据变成两位一组的字符串，根据这个规则编写编码函数encode如下：

```
String encode(String s)
{
    if(s==null) return s;
    byte[] buf=Encoding.UTF8.GetBytes(s);
    s="";
    foreach(byte b in buf) s=s+b.ToString("X2");
    return s;
}
```

例如，"A&B"的字符串编码为"412642"。如果要解码，就把字符串每两个字符为一组取出来转为二进制数据，再通过UTF8编码规则转为字符串即可，解码函数decode如下：

```
String decode(String s)
{
    if(s==null) return s;
    byte[] buf=new byte[s.Length / 2];
    for (int i=0; i<buf.Length; i++)
        buf[i] = byte.Parse(s.Substring(2 * i, 2), System.Globalization.
        NumberStyles.HexNumber);
    return Encoding.UTF8.GetString(buf);
}
```

执行decode("412642")就是"A&B"。有了这样的编码与解码函数，用户信息的传递可以改为

```
url+"?uName="+encode(uName)+"&uPass="+decode(uPass)
```

服务器获取uName与uPass后要用decode进行解码：

```
String uName=decode(context.Response.QueryString["uName"]);
String uPass=decode(context.Response.QueryString["uPass"]);
```

由于编码后的字符串中只包含通常的数字与A～F的几个英文字符，因此传递保证是安全的。

2. 结果返回

如果要知道服务器是否正确获取了客户端提交的用户信息，可以设计一个信息类来

返回服务器的结果，这个类 ResultClass 如下：

```
public class ResultClass
{
    public String state { get; set; }
    public String message { get; set; }
}
```

其中 state 是结果的状态，如果操作成功则 state 为 success，否则为 error，而 message 为操作结果的字符串，本程序用于提交用户的姓名与密码信息。

由于客户端采用 WebClient 的异步函数 DownloadStringAsync 来提交信息，因此服务器返回结果时会触发预先设置好的异步回调函数 client_DownloadStringCompleted，在这个函数中就可以通过 e.Result 获取服务器的返回值。

2.1.3 服务器程序

创建服务器程序的过程如下：

(1) 启动 Visual Studio，新建一个项目，选择“其他类型项目”中的“Visual Studio 解决方案”，然后再次选择“空白解决方案”，选择一个工作目录(例如 C:\book)，输入解决方案的名称(例如 ImageServer)，如图 2-7 所示。选择好后单击“确定”按钮，在解决方案管理器中就可以看到 ImageServer 的解决方案。

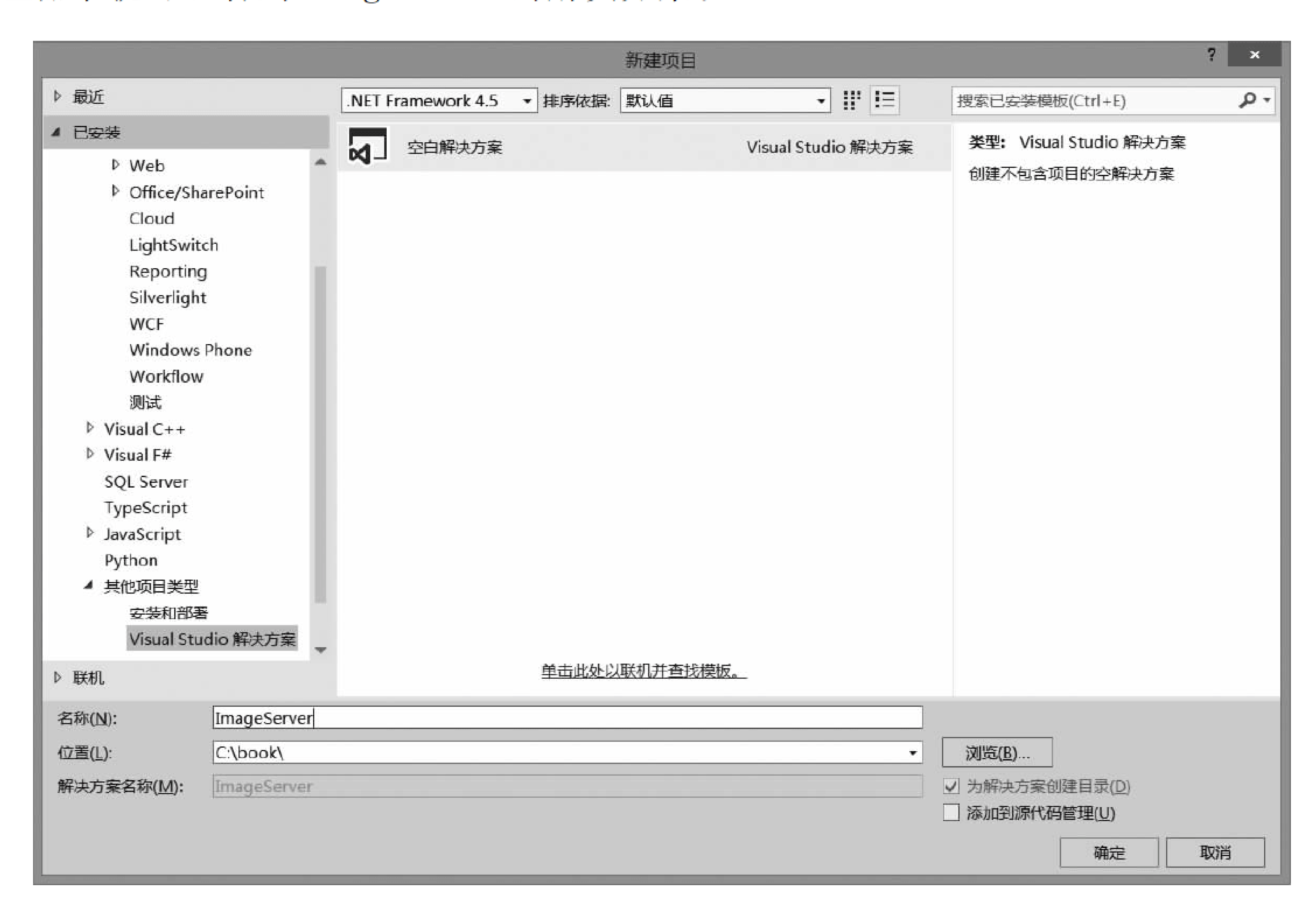


图 2-7 建立解决方案

(2) 由于序列化与反序列化函数、编码与反编码函数等要反复使用，为了简单起见，可以把这些函数做成一个 DLL 动态库。右击解决方案资源管理器中的“解决方案

ImageServer”,弹出快捷菜单,选择“添加”→“新建项目”,弹出“添加新项目”对话框,选择“类库”,在名称中输入 MyDLL,如图 2-8 所示。确定后就在解决方案中增加了一个 MyDLL 的项目。

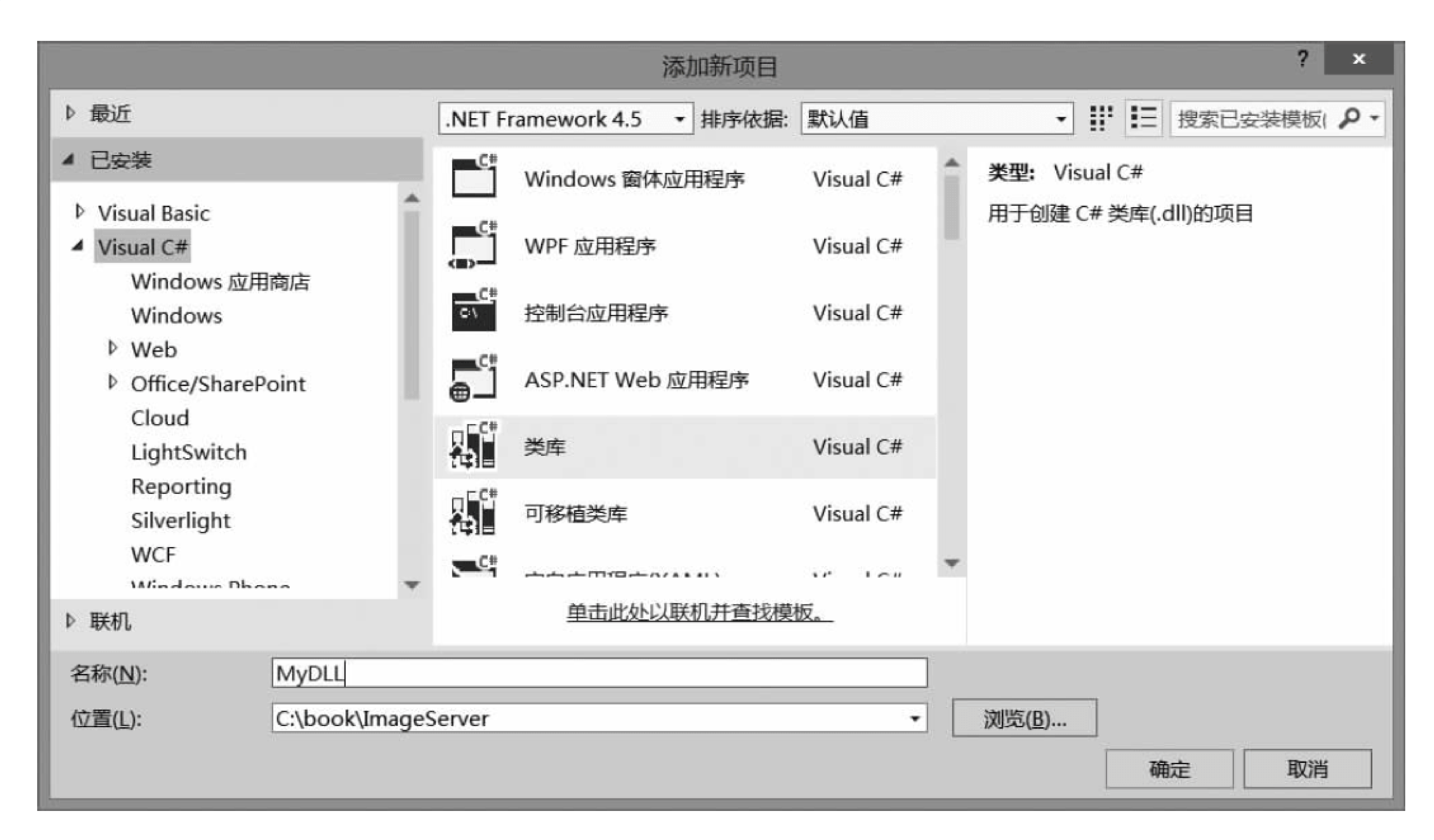


图 2-8 建立 MyDLL 库

(3) 再次右击 MyDLL,弹出快捷菜单,选择“添加”→“类”命令,在 MyDLL 中建立一个类 My,在这个类中封装了 encode 编码函数与 decode 解码函数,同时还封装了 serialize 的 XML 序列化函数与 deserialize 的反序列化函数,My 类设计如下:

```
namespace MyDLL
{
    public class My
    {
        public static String encode(String s)
        {
            if(s==null) return s;
            byte[] buf=Encoding.UTF8.GetBytes(s);
            s="";
            foreach (byte b in buf) s=s+b.ToString("X2");
            return s;
        }
        public static String decode(String s)
        {
            if(s==null) return s;
            byte[] buf=new byte[s.Length / 2];
            for (int i=0; i<buf.Length; i++)
                buf[i]=byte.Parse(s.Substring(2 * i, 2), System.Globalization.
                NumberStyles.HexNumber);
```

```
                return Encoding.UTF8.GetString(buf);
            }
            public static String serialize<T>(T obj)
            {
                XmlSerializer xml=new XmlSerializer(typeof(T));
                MemoryStream ms=new MemoryStream();
                xml.Serialize(ms, obj);
                String s=Encoding.UTF8.GetString(ms.ToArray());
                return s;
            }
            public static T deserialize<T>(String s)
            {
                XmlSerializer xml=new XmlSerializer(typeof(T));
                byte[] buf=Encoding.UTF8.GetBytes(s);
                MemoryStream ms=new MemoryStream(buf);
                T obj=(T)xml.Deserialize(ms);
                return obj;
            }
        }
    }
```

(4) 返回信息类 ResultClass 是服务器与客户端传递信息的一个重要结构，为了实现对它的封装，仿照建立 MyDLL 与 My 类的方法，在解决方案管理器中再次建立一个名称为 Model 的类库，在这个类库中建立 ResultClass 类如下：

```
namespace Model
{
    public class ResultClass
    {
        public String state { get; set; }
        public String message { get; set; }
    }
}
```

(5) 再次右击"解决方案 ImageServer"，弹出快捷菜单，选择"添加"→"现有网站"命令，将本书默认的网站 D:\web 添加到项目中。

(6) 再次右击 web 网站，弹出快捷菜单，选择"添加新项"命令，弹出对话框，选择"一般处理程序"，在该网站下增加一个名称为 ImageServer. ashx 的程序，处理完后，服务器中的解决方案资源管理器如图 2-9 所示。



图 2-9 服务器程序结构

(7) 为了让 web 中的 ImageServer. ashx 使用 My 类与 ResultClass 类，再次右击 web 网站，弹出快捷菜

单，选择“添加”→“引用”命令，弹出“引用管理器”对话框，选择“项目”，勾选 MyDLL 和 Model 后确定，如图 2-10 所示。

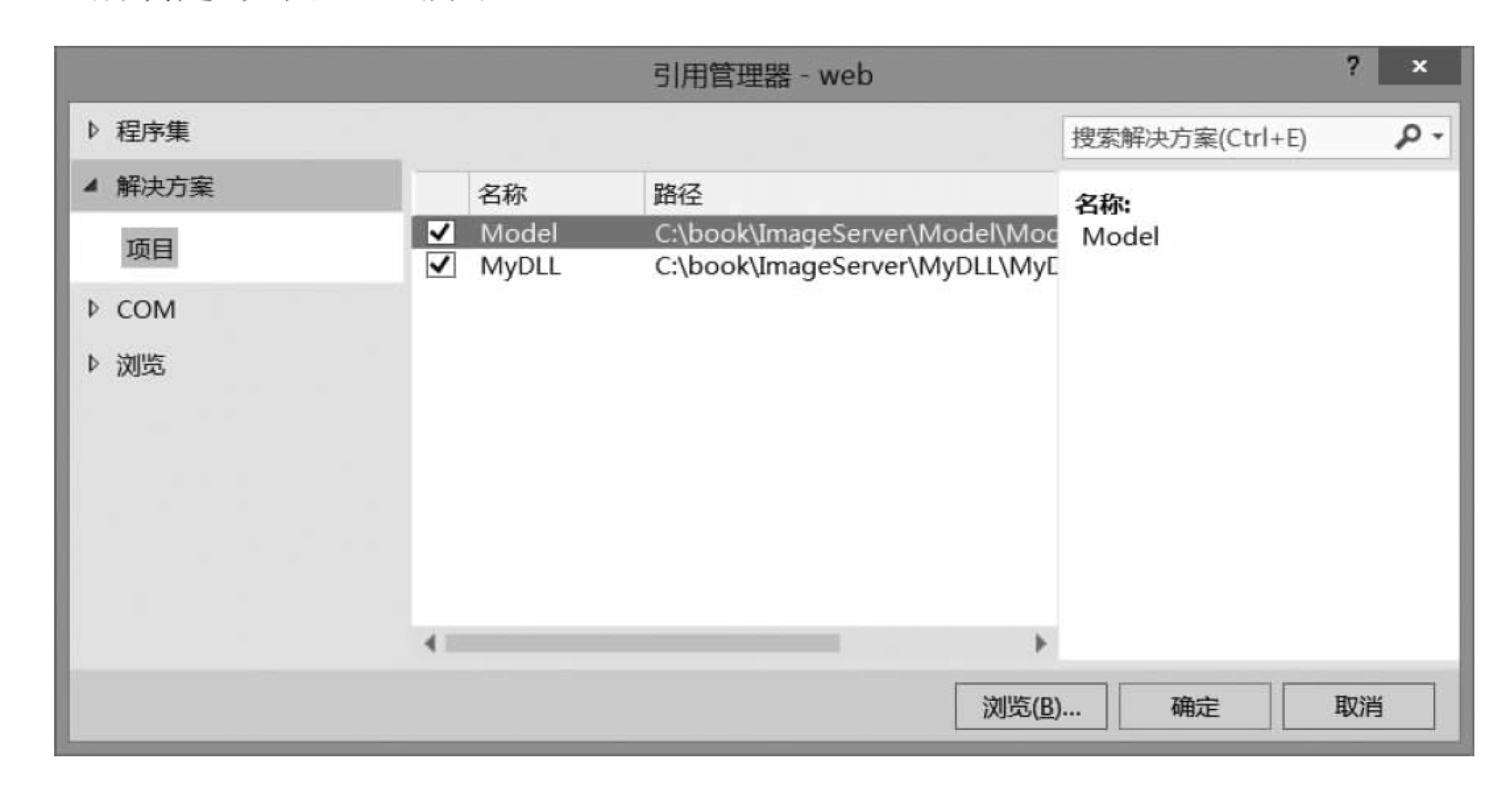


图 2-10　添加引用

然后在 ImageServer. ashx 程序的开头部分写上下列两句，这样 ImageServer. ashx 中就可以使用 My 与 ResultClass 类了：

```
using MyDLL;
using Model;
```

(8) 编写 ImageServer. ashx 程序，该程序读取客户端传递来的 uName 与 uPass 用户名称与密码，然后再返回给客户端，程序如下：

```
<%@WebHandler Language="C# " Class="ImageServer" %>
…//省略 using 的部分代码
using MyDLL;
using Model;
public class ImageServer : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        ResultClass res=new ResultClass();
        try
        {
            String uName=My.decode(context.Request.QueryString["uName"]);
            String uPass=My.decode(context.Request.QueryString["uPass"]);
            res.state="success";
            res.message=uName+","+uPass;
        }
        catch(Exception exp)
        {
```

```
                res.state="error"; res.message=exp.Message;
            }
            context.Response.ContentType="text/plain";
            context.Response.Write(My.serialize<ResultClass>(res));
            context.Response.Flush();
        }
        public bool IsReusable
        {
            get{ return false; }
        }
    }
```

(9) 全部程序编写好后，执行“重新生成解决方案”命令，就可以看到在 web 下建立了一个 Bin 的文件夹，而且生成的 MyDLL. dll 与 Model. dll 动态库就存放在这个文件夹中。在浏览器中输入 http://localhost/web/ImageServer. ashx 地址进行测试，可以看到图 2-11 所示的结果。



图 2-11　测试结果

2.1.4　客户端程序

客户端程序 ImageClient 设计成 WPF 窗体程序，主要包含界面设计与程序代码设计两个部分。

1. 界面设计

客户端的界面很简单，在 WPF 程序的窗体上放一个名为 txtUser 的文本框用于输入用户名，一个名为 txtPass 的密码框用于输入密码，一个名称为 txtMsg 的 TextBlock，再放一个名为 btLogin 的按钮 Button 实现注册与登录，主要 XAML 代码如下：

```
<TextBox x:Name="txtUser" Text="xxx" HorizontalAlignment="Left" Height=
"23" Margin="42,15,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width=
"50"/>
<PasswordBox x:Name="txtPass" Password="123"  HorizontalAlignment="Left"
VerticalAlignment="Top" Margin="42,43,0,0" Width="50"/>
<Button x:Name="btLogin" Content="提交" HorizontalAlignment="Left"
VerticalAlignment="Top" Width="49" Margin="10,67,0,0" Click="btLogin_
```

```
Click"/>
< TextBlock HorizontalAlignment =" Left " Text =" 结 果 " x: Name =" txtMsg "
TextWrapping="Wrap"  VerticalAlignment="Top" Margin="68,71,0,0"/>
```

2. 程序设计

客户端采用 WPF 窗体结构，主要过程如下：

(1) 执行 Visual Studio 中的“文件”→“新建项目”命令，弹出项目对话框，选择“WPF 窗体程序”，输入名称 ImageClient 后确定，就建立了一个 ImageClient 的解决方案与项目。

(2) 在解决方案资源管理器中右击“解决方案 ImageClient”，弹出快捷菜单，选择“添加”→“新建项”命令，弹出对话框，选择“类库”，输入名称 MyDLL 就建立了 MyDLL 的项目。在 MyDLL 中再建立一个 My 的类，结构与服务器完全一样。

(3) 在解决方案资源管理器中右击“解决方案 ImageClient”，弹出快捷菜单，选择“添加”→“新建项”命令，弹出对话框，选择“类库”，输入名称 Model 就建立了 Model 的项目。在 Model 中再建立一个 ResultClass 的类，结构与服务器完全一样。客户端建立了 MyDLL 与 Model 后的解决方案资源管理器结构如图 2-12 所示。



图 2-12 客户端程序结构

(4) 右击 ImageClient 项目，弹出快捷菜单，选择“添加”→“引用”命令，弹出对话框，与服务器一样引用 MyDLL 与 Model，然后在 ImageClient 中的 MainWindow. xaml. cs 程序开始部分写上两条引用语句：

```
using MyDLL;
using Model;
```

(5) 编写程序 MainWindow. xaml. cs，根据前面的分析，客户端程序设计如下(同样限于篇幅省去了 using System 等引用部分)，重要的部分都做了注释。

```
…//省略 using 的部分代码
using MyDLL;
using Model;
namespace ImageClient
{
    public partial class MainWindow : Window
    {
        WebClient client;
        String url="http://localhost/web/ImageServer.ashx";
        public MainWindow()
        {
            InitializeComponent();
```

```
            client=new WebClient();
            client.Encoding=Encoding.UTF8;
            //设置异步回调函数
            client.DownloadStringCompleted+=client_DownloadStringCompleted;
        }
        void showMsg(String s)
        {
            MessageBox.Show(s, "Information", MessageBoxButton.OK);
        }
        void client_DownloadStringCompleted(object sender, DownloadString-
        CompletedEventArgs e)
        {
            if(e.Error==null)
            {
                ResultClass res=My.deserialize<ResultClass>(e.Result);
                if(res.state=="success")
                {
                    txtMsg.Text=res.message;
                }
                else showMsg(res.message);
            }
            else  showMsg(e.Error.Message);
        }
        private void btLogin_Click(object sender, RoutedEventArgs e)
        {
            String uName=My.encode(txtUser.Text.Trim());
            String uPass=My.encode(txtPass.Password.Trim());
            try
            {
                Uri uri=new Uri(url+"?uName="+uName+"&uPass="+uPass,
                UriKind.Absolute);
                client.DownloadStringAsync(uri);
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
}
```

(6) 全部程序编写好后，执行"重新生成解决方案"命令，就生成了 ImageClient. exe 执行程序，而且可以看到生成的 MyDLL. dll 与 Model. dll 动态库就存放在 ImageClient. exe 所在的这个文件夹中。执行客户端程序，结果如图 2-6 所示。

2.1.5 拓展训练

实际上还有一种传输用户信息的方法，就是使用 WebClient 的 UploadValues 函数，

该函数原型如下：

```
public byte[]  UploadValues(Uri uri,String method,NameValueCollection nvc)
```

其中 nvc 是一个名字/值对。这个函数在执行时会自动把 HTTP 协议中的 ContentType 设置为 application/x-www-form-urlencoded，即用表单的形式来传输，因此可以很方便地传输各种值，并且不用对值进行编码。

使用这个方法客户信息可以这样上传：

```
void btLogin_Click(object sender, RoutedEventArgs e)
{
    String uName=txtUser.Text.Trim();
    String uPass=txtPass.Password.Trim();
    try
    {
        Uri uri=new Uri(url, UriKind.Absolute);
        NameValueCollection nvc=new NameValueCollection();
        nvc.Add("uName", uName);
        nvc.Add("uPass", uPass);
        client.UploadValuesAsync(uri, "POST", nvc);
    }
    catch(Exception exp) { showMsg(exp.Message); }
}
```

由于是表单传输，所以在服务器一端可以使用 context.Request.Form 的形式来接收各个值：

```
public void ProcessRequest(HttpContext context)
{
    ResultClass res=new ResultClass();
    try
    {
        String uName=context.Request.Form["uName"];
        String uPass=context.Request.Form["uPass"];
        res.state="success";
        res.message=uName+","+uPass;
    }
    catch(Exception exp)
    {
        res.state="error"; res.message=exp.Message;
    }
    context.Response.ContentType="text/plain";
    context.Response.Write(My.serialize<ResultClass>(res));
    context.Response.Flush();
}
```

其他函数不用变化，修改后的程序效果一样。

2.2 用户注册与登录

2.2.1 案例展示

设计服务器程序管理用户的登录或者注册，设计客户端完成用户的注册或者登录。如果是新用户，即所输入的用户名在服务器中不存在，则用这个用户名执行注册；如果用户已经存在，则直接登录，如图 2-13 所示。



图 2-13 用户注册与登录

2.2.2 技术要点

1. 数据库设计

在服务器的 SQL Server 中建立数据库 NetImages，然后在数据库中建立 users 表。users 表是用户信息表，有两个字段：一个是 uName，为用户名称，这是表的关键字段；另外一个是 uPass，为用户密码。执行 SQL 命令：

```
create table users
(
    uName varchar(32) primary key,
    uPass varchar(512)
)
```

2. 数据库操作

按照 3 层结构习惯的编程方法，把对数据库的操作放在数据访问层(DAL 层)，具体做法如下：

(1) 在解决方案管理器右击解决方案 ImageServer，弹出快捷菜单，选择“添加新项”命令，弹出添加新项对话框，再次选择“类库”，在名称中输入 DAL 后确定。完成后，在解决方案管理器中可以看到一个 DAL 的项目。

(2) 在解决方案管理器中右击 DAL，弹出快捷菜单，选择“添加类”命令，在 DAL 中增加一个名称为 DBHelper 的类。DBHelper 类是访问和执行数据库命令的基本类，它包含了常用的数据库操作，设计如下：

```
namespace DAL
{
    public class DBHelper
    {
        public static String conString="Data Source=localhost;
        Initial Catalog=NetImages; Integrated Security=SSPI;
        MultipleActiveResultSets=true";
```

```
private SqlConnection con;
private SqlCommand cmd;
public DBHelper(SqlConnection con)
{
    con.Open();
    this.con=con;
    cmd=new SqlCommand();
    cmd.Connection=con;
}
public int executeCommand(String sql)
{
    cmd.CommandText=sql;
    cmd.Parameters.Clear();
    return cmd.ExecuteNonQuery();
}
public int executeCommand(String sql,params SqlParameter[] values)
{
    cmd.CommandText=sql;
    cmd.Parameters.Clear();
    cmd.Parameters.AddRange(values);
    return cmd.ExecuteNonQuery();
}
public SqlDataReader getReader(String sql)
{
    cmd.CommandText=sql;
    cmd.Parameters.Clear();
    SqlDataReader reader=cmd.ExecuteReader();
    return reader;
}
public SqlDataReader getReader(String sql, params SqlParameter[] values)
{
    cmd.CommandText=sql;
    cmd.Parameters.Clear();
    cmd.Parameters.AddRange(values);
    SqlDataReader reader=cmd.ExecuteReader();
    return reader;
}
public SqlDataAdapter getAdapter(String sql)
{
    cmd.CommandText=sql;
    cmd.Parameters.Clear();
    SqlDataAdapter adapter=new SqlDataAdapter(cmd);
    return adapter;
}
```

```
        public SqlDataAdapter getAdapter(String sql, params SqlParameter[]
        values)
        {
            cmd.CommandText=sql;
            cmd.Parameters.Clear();
            cmd.Parameters.AddRange(values);
            SqlDataAdapter adapter=new SqlDataAdapter(cmd);
            return adapter;
        }
        public int getScalar(String sql)
        {
            cmd.CommandText=sql;
            cmd.Parameters.Clear();
            return(int)cmd.ExecuteScalar();
        }
        public int getScalar(String sql, params SqlParameter[] values)
        {
            cmd.CommandText=sql;
            cmd.Parameters.Clear();
            cmd.Parameters.AddRange(values);
            return(int)cmd.ExecuteScalar();
        }
    }
}
```

DBHelper 中 conString 是连接数据库的字符串,类中主要包括的函数有执行 SQL 命令的 executeCommand 函数、获取数据阅读器 SqlDataReader 的 getReader 函数、获取数据适配器 SqlDataAdapter 的 getAdapter 函数、获取单值的 getScalar 函数。每个函数有两个重载形式,一个不带参数,另一个可以带任意多个参数。

3. 用户信息

用户注册与登录时提交用户名称 uName 与用户密码 uPass,除此之外还要提交一个信息给服务器表示执行用户注册登录操作。这个信息约定用 opt 参数的值来表示,当 opt 值为 loginUser 时通知服务器执行用户注册登录。

客户端通过 WebClient 的 DownloadString 函数完成字符串的上传,主要代码如下:

```
String param="uName="+My.encode(uName)+"&uPass="+My.encode(uPass);
Uri uri=new Uri(url+"?opt=loginUser&"+param, UriKind.Absolute);
client.DonloadStringAsync(uri);
```

其中 opt="loginUser"这个参数通过地址传输给服务器,服务器检测这个 opt 参数以便做出相应的登录操作。

4. 用户注册登录

服务器接收到客户端上传来的用户信息后，就把该用户信息(uName 和 uPass)插入到数据库表 users 中，由于 uName 是 users 的关键字段，如果插入成功，说明该用户原来不存在，完成注册。如果插入不成功，说明该用户已经存在，这时再次比较密码是否正确，如果正确就完成登录，如果不正确就登录失败。

客户端提交用户名称 uName 与密码 uPass 后，这些信息传递给 DAL 层就可以完成注册登录的操作。在 DAL 再次建立一个 UserService 类，完成注册登录的操作，程序如下：

```
namespace DAL
{
    public class UserService
    {
        public String login(String uName, String uPass)
        {
            String s="failed";
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                  SqlParameter pName=new SqlParameter("@uName", SqlDbType.
                  Char); pName.Value=uName;
                  SqlParameter pPass=new SqlParameter("@uPass", SqlDbType.
                  Char); pPass.Value=uPass;
                try
                {
                    DB.executeCommand("insert into users values (@uName, @
                   uPass)", pName, pPass);
                    s="registered";
                }
                catch
                {
                    if(DB.getScalar("select count(*) from users where uName=
                   @uName and uPass=@uPass", pName, pPass)>0) s="logined"; ;
                }
                con.Close();
            }
            return s;
        }
}
```

5. 结果返回

服务器端完成用户注册登录后要返回一个结果给客户端，告诉客户端有没有正确完

成这些操作，是完成了注册还是登录操作，可以在 Model 中设计一个结果信息类 ResultClass 来表示这些结果，这个类如下：

```
public class ResultClass
{
    public String state { get; set; }
    public String message { get; set; }
}
```

其中 state 是结果的状态，如果操作成功则 state 为 success，否则为 error。而 message 为操作结果的字符串，注册成功为 registered，登录成功为 logined，失败为 failed。

2.2.3 服务器程序

服务器程序网站 Web 引用 DAL、MyDLL 与 Model，全部建立好后，服务器中的解决方案资源管理器结构如图 2-14 所示。



图 2-14 服务器解决方案

根据前面的分析设计 ImageServer. ashx 服务器，主要代码如下(限于篇幅，省去了 using System 等的引用部分)：

```
…//省略 using 的引用代码部分
using Model;
using MyDLL;
using DAL;
public class ImageServer: IHttpHandler
{
    HttpContext context;
    String loginUser()
    {
        //用户注册与登录
        String uName=My.decode(context.Request.QueryString["uName"]);
        String uPass=My.decode(context.Request.QueryString["uPass"]);
        UserService userService=new UserService();
        return userService.login(uName, uPass);
    }
    public void ProcessRequest(HttpContext context)
    {
        this.context=context;
        ResultClass res=new ResultClass { state="success" };
        String opt=context.Request["opt"];
        try
```

```
            {
                if(opt=="loginUser") res.message=loginUser();
            }
            catch(Exception exp)
            {
                //错误信息
                res.state="error"; res.message=exp.Message;
            }
            //返回 result 的 XML 文本信息
            context.Response.Clear();
            context.Response.ContentType="text/plain";
            context.Response.Write(My.serialize<ResultClass>(res));
            context.Response.Flush();
        }
        public bool IsReusable {
            get { return false; }
        }
    }
```

2.2.4 客户端程序

客户端程序 ImageClient 设计成 WPF 窗体程序，主要包含界面设计与程序代码设计两个部分。界面与 2.1 节中的几乎完全一样。根据前面的分析，客户端程序设计如下(同样限于篇幅省去了 using System 等引用部分)，重要的部分都做了注释。

```
    …//省略 using 的引用代码部分
    using MyDLL;
    using Model;
    namespace ImageClient
    {
        public partial class MainWindow : Window
        {
            WebClient client;
            String url="http://localhost/web/ImageServer.ashx";
            public MainWindow()
            {
                InitializeComponent();
                client=new WebClient();
                client.Encoding=Encoding.UTF8;
                //设置异步函数
                client.DownloadStringCompleted+=client_DownloadStringCompleted;
             }
            void showMsg(String s)
            {
                MessageBox.Show(s, "Information", MessageBoxButton.OK);
```

```
        }
        void client_DownloadStringCompleted(object sender,
        DownloadStringCompletedEventArgs e)
        {
            //用户注册登录触发此函数
            if(e.Error==null)
            {
                ResultClass res=My.deserialize<ResultClass>(e.Result);
                if(res.state=="success") txtMsg.Text=res.message;
                else showMsg(res.message);
            }
            else  showMsg(e.Error.Message);
        }
        private void btLogin_Click(object sender, RoutedEventArgs e)
        {
            String uName=My.encode(txtUser.Text.Trim());
            String uPass=My.encode(txtPass.Password.Trim());
            try
            {
                Uri uri=new Uri(url+"?opt=loginUser&uName="+uName+"&uPass="
                +uPass, UriKind.Absolute);
                client.DownloadStringAsync(uri);
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
}
```

在 client_DownloadStringCompleted 函数中使用 e. Error 来判断服务器的错误情况，如果访问的服务器网址存在，使得 ImageServer. ashx 工作，那么就不会有错误，e. Error 为 null。如果 ImageServer. ashx 不工作，e. Error 就会不为 null，此时 e. Error. Message 给出错误信息。如果网址是正确的而且 ImageServer. ashx 正常工作，由于数据库等原因造成错误，那么服务器会抛出异常，这个异常错误在返回的信息 res. message 中会体现。

2.2.5 拓展训练

客户端登录时提交用户名称与密码，实际应用中密码在提交给服务器时一般做加密处理，以防止密码在传输的过程中被窃取。一种常用的加密方法是 MD5 加密法，具体来说就是把用户输入的密码进行 MD5 加密计算变成另外一个字符串，把用户名称与该加密字符串发送给服务器，服务器接收后从数据库中取出用户密码的加密字符串进行比较来判断该用户的合法性。经过 MD5 加密的字符串就算被窃取，也不可能解密出原密码字符串，这样的登录过程就比较安全了。

MD5 加密可以使用 MD5 类完成，其中通过 MD5CryptoServiceProvider 类建立一个

MD5 对象，调用该对象的 ComputeHash 函数可以得到一个二进制数组的哈希值，这个哈希值转为十六进制字符串就是加密字符串，主要程序如下：

```
String encryptString(String s)
{
    MD5 md5=new MD5CryptoServiceProvider();
    byte[] buf=Encoding.UTF8.GetBytes(s);
    buf=md5.ComputeHash(buf);
    s="";
    foreach (byte x in buf) s=s+x.ToString("X2");
    return s;
}
```

在使用了 MD5 加密后，客户注册登录函数修改如下：

```
private void btLogin_Click(object sender, RoutedEventArgs e)
{
    String uName=My.encode(txtUser.Text.Trim());
    String uPass=My.encode(encryptString(txtPass.Password.Trim()));
    try
    {
         Uri uri = new Uri (url + "?opt= loginUser&uName=" + uName + "&uPass=" +
        uPass, UriKind.Absolute);
        client.DownloadStringAsync(uri);
    }
    catch(Exception exp) { showMsg(exp.Message); }
}
```

这样客户端在注册登录时向服务器提交的是加密后的密码字符串，而不是原始明文的密码。

2.3 图片上传与存储

2.3.1 案例展示

设计服务器与客户端程序实现图片的上传，客户端单击“图片上传”按钮后选择一张图片上传给服务器，服务器接收此图片并存储到数据库中，同时返回该图片的 ID 号给客户端，如图 2-15 所示。

图 2-15　图片上传

2.3.2 技术要点

1. 图片数据库表

首先在数据库中建立一个存储图片的表 images，SQL 的建表命令如下：

```
create table images
(
    ID int identity(1,1) primary key,
    uName varchar(32) foreign key references users(uName),
    iDate varchar(32),
    iFile varchar(256),
    iShared bit not null default 0,
    iData image
)
```

其中各个字段的含义如表 2-1 所示。

表 2-1 Images 表字段

字段名称	备　　注
ID	自动增长列,主键,图片的 ID 号
uName	用户名称,说明该图片归该用户所有
iDate	存储图片的日期时间
iFile	图片的名称
iShared	图片是否共享,值为 1 时共享,为 0 时不共享
iDate	图片的二进制数据

2. 客户端图片上传

客户端上传图片时,首先选择要上传的图片,获取图片的名称与二进制数据,确定图片的所有者,然后把这些数据上传给服务器。WebClient 有一个二进制数据上传函数 UploadData,方法原型是:

```
public byte[] UploadData(Uri uri ,Sting method,byte[] data)
```

其中 uri 为要上传的地址,method 默认为 POST,data 为要上传的二进制数据。在上传图片时,除了要上传图片本身的二进制数据外,还要上传图片的名称 iFile 与所有者 uName 等信息,可以把 iFile、uName 这些信息附加在地址栏上,在附加时也同样对它们进行编码,即 Uri 地址可以这样创建:

```
Uri uri=new Uri(url+"?opt=uploadImage&uName="+My.encode(uName)+"&uPass="
+My.encode(uPass)+"&iFile="+My.encode(fn), UriKind.Absolute);
```

其中 opt=uploadImage 告诉服务器执行上传图片的操作,uName、uPass、iFile 是用户名称、密码、文件名称,它们的值都需要进行编码。

3. 服务器图片存储

客户端上传图片后,服务器通过 Request 对象的 QueryString 获取用户名称 uName

与图片名称 iFile，通过 BinaryRead 获取图片的二进制数据 data：

```
String uName=My.decode(context.Request.QueryString["uName"]);
String iFile=My.decode(context.Request.QueryString["iFile"]);
byte[] data=context.Request.BinaryRead(context.Request.TotalBytes);
```

在 DAL 中定义一个 ImageService 类，该类中有一个 uploadImage 函数，实现图片的存储，这个函数如下：

```
namespace DAL
{
    public class ImageService
    {
      bool verifyUser(DBHelper DB,String uName, String uPass)
        {
            SqlParameter pName=new SqlParameter("@uName", SqlDbType.Char);
            pName.Value=uName;
            SqlParameter pPass=new SqlParameter("@uPass", SqlDbType.Char);
            pPass.Value=uPass;
            return((int)DB.getScalar("select count(*) from users where uName
            =@uName and uPass=@uPass", pName, pPass)>0);
        }
        public String uploadImage(String uName,String uPass,String iFile,
        byte[] data)
        {
            String ID="0";
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                if(verifyUser(DB,uName, uPass))
                {
                    SqlParameter pName=new SqlParameter("@uName", SqlDbType.
                    Char); pName.Value=uName;
                    SqlParameter pDate=new SqlParameter("@iDate", SqlDbType.
                    Char); pDate.Value=DateTime.Now.ToString("yyyy-MM-dd HH:
                    mm:ss");
                    SqlParameter pFile=new SqlParameter("@iFile", SqlDbType.
                    Char); pFile.Value=iFile;
                    SqlParameter pData=new SqlParameter("@iData", SqlDbType.
                    Image); pData.Value=data;
                    if(DB.executeCommand("insert into images (uName, iDate,
                    iFile,iData) values (@uName, @iDate, @iFile, @iData)",
                    pName, pDate, pFile, pData)>0)
                    {
                        //插入成功时获取插入记录的 ID
```

```
                        ID=DB.getScalar("select top 1 ID from images order by ID
                        desc").ToString();
                    }
                }
                con.Close();
            }
            return ID;
        }
    }
```

在图片上传时要先调用 verifyUser 对用户进行验证，只有注册的用户才能上传图片。图片上传后在数据库表中增加一条新记录，把 uName、iFile、data 等存储到数据库表中，并返回新增加记录的 ID 号。

4. 结果的返回

服务器存储图片后，images 表增加一条记录，这条记录的 ID 号是数据库自动计算设置的，因此程序要读取该 ID 值，然后返回给客户端，客户端就凭这个 ID 值来查找服务器的图片。

2.3.3 服务器程序

服务器端的主要函数是 UploadImage 函数，该函数在 images 表中增加一条记录，并返回新记录的 ID 值。

```
using Model;
using MyDLL;
using DAL;
public class ImageServer: IHttpHandler
{
    HttpContext context;
    String uploadImage()
    {
        //上传图片
        String uName=My.decode(context.Request.QueryString["uName"]);
        String uPass=My.decode(context.Request.QueryString["uPass"]);
        String iFile=My.decode(context.Request.QueryString["iFile"]);
        byte[] data=context.Request.BinaryRead(context.Request.TotalBytes);
        ImageService imageService=new ImageService();
        return imageService.uploadImage(uName,uPass,iFile, data);
    }
    public void ProcessRequest(HttpContext context)
    {
        this.context=context;
        ResultClass res=new ResultClass { state="success" };
```

```
            String opt=context.Request["opt"];
            try
            {
                if(opt=="uploadImage")  res.message=uploadImage();
            }
            catch(Exception exp)
            {
                //错误信息
                res.state="error";
                res.message=exp.Message;
            }
            //返回 result 的 XML 文本信息
            context.Response.Clear();
            context.Response.ContentType="text/plain";
            context.Response.Write(My.serialize<ResultClass>(res));
            context.Response.Flush();
        }
        public bool IsReusable {
            get { return false; }
        }
    }
```

2.3.4 客户端程序

1. 界面设计

客户端的界面很简单，在窗体上放一个名为 btUpload 的按钮 Button，再放一个名为 txtMsg 的文本标签 TextBlock 即可。

```
<Grid>
      <TextBox x:Name="txtUser" Text="xxx" HorizontalAlignment="Left"
      Height="23" Margin="42,15,0,0" TextWrapping="Wrap" VerticalAlignment
      ="Top" Width="50"/>
      <Button x:Name="btUpload" Content="图片上传" HorizontalAlignment=
      "Left" VerticalAlignment="Top" Width="66" Margin="106,29,0,0" Click=
     "btUpload_Click"/>
      <TextBlock HorizontalAlignment="Left" Text="结果" x:Name="txtMsg"
      TextWrapping="Wrap" VerticalAlignment="Top" Margin="10,68,0,0"/>
     <TextBlock HorizontalAlignment="Left" Text="用户" TextWrapping="Wrap"
       VerticalAlignment="Top" Margin="10,21,0,0" RenderTransformOrigin=
     "0.583,-1.105"/>
     <PasswordBox x:Name="txtPass" HorizontalAlignment="Left"
     VerticalAlignment="Top" Width="50" Password="123"
     RenderTransformOrigin="1.207,3.818" Margin="42,43,0,0"/>
```

```
    <TextBlock HorizontalAlignment="Left" Text="密码"  TextWrapping=
    "Wrap"  VerticalAlignment="Top" Margin="10,48,0,0"/>
</Grid>
```

2. 程序设计

客户端程序要选择本地磁盘中的一个文件，要用到打开文件对话框，因此要在程序头部写上语句：

```
using Microsoft.Win32;
```

主要的程序部分如下：

```
   …//省略部分 using 代码
using Microsoft.Win32;
using MyDLL;
using Model;
namespace ImageClient
{
    public partial class MainWindow : Window
    {
        WebClient client;
        String url="http://localhost/web/ImageServer.ashx";
        public MainWindow()
        {
            InitializeComponent();
            client=new WebClient();
            client.Encoding=Encoding.UTF8;
            //设置异步回调函数
            client.UploadDataCompleted+=client_UploadDataCompleted;
        }
        void client_UploadDataCompleted(object sender,
        UploadDataCompletedEventArgs e)
        {
            if(e.Error==null)
            {
                //上传完成后
                String s=Encoding.UTF8.GetString(e.Result);
                ResultClass res=My.deserialize<ResultClass>(s);
                if(res.state=="success")
                {
                    if(res.message !="0")
                    {
                        txtMsg.Text="图片上传成功！ID="+res.message;
                    }
```

```
                    else showMsg("图片上传失败！");
                }
                else showMsg(res.message);
            }
            else  showMsg(e.Error.Message);
        }
        void showMsg(String s)
        {
            MessageBox.Show(s, "Information", MessageBoxButton.OK);
        }
        String encryptString(String s)
        {
            MD5 md5=new MD5CryptoServiceProvider();
            byte[] buf=Encoding.UTF8.GetBytes(s);
            buf=md5.ComputeHash(buf);
            s="";
            foreach (byte x in buf) s=s+x.ToString("X2");
            return s;
        }
        private void btUpload_Click(object sender, RoutedEventArgs e)
        {
            String uName=txtUser.Text.Trim();
            String uPass=txtPass.Password.Trim();
            OpenFileDialog dlg=new OpenFileDialog();
            dlg.Filter="Images|*.jpg;*.png";
            if((bool)dlg.ShowDialog() && uName !="" && uPass!="")
            {
                try
                {
                    String fn=dlg.FileName;
                    int p=fn.LastIndexOf("\\");
                    if(p>=0) fn=fn.Substring(p+1);
                    FileStream fs=new FileStream(dlg.FileName, FileMode.Open);
                    byte[] data=new byte[fs.Length];
                    fs.Read(data, 0, data.Length);
                    fs.Close();
                    uPass=encryptString(uPass);
                    Uri uri=new Uri(url+"?opt=uploadImage&uName="+My.encode
                    (uName)+"&uPass="+My.encode(uPass)+"&iFile="+My.encode
                    (fn), UriKind.Absolute);
                    //上传二进制数组 data
                    client.UploadDataAsync(uri, "POST", data);
                }
                catch(Exception exp) { showMsg(exp.Message); }
            }
```

```
            }
        }
    }
```

2.3.5 拓展训练

实际上图片包含ID号、所有者uName、图片文件名称iFile、存储日期iDate、图片二进制数据iData等信息，为了很好地表示一张图片，在客户端与服务器中的Model中建立图片信息类ImageClass来表示一张图片的所有这些数据：

```
public class ImageClass
{
    public int ID { get; set; }
    public String uName { get; set; }
    public String iDate { get; set; }
    public String iFile { get; set; }
    public byte[] iData { get; set; }
}
```

客户端上传图片时，可以把图片所有者uName、图片名称iFile信息以及图片数据一起组合成一个ImageClass类，然后把这个类序列化成XML文本字符串，由于iData是一个二进制数组，在序列化时会被转为Base64的字符串，因此这个XML字符串很长。

客户端可以使用WebClient的UploadString函数来上传，服务器接收到这个字符串后再反序列化回ImageClass对象，就可以存储到数据库里了。

2.4 图片共享与设置

2.4.1 案例展示

客户端使用DataGrid控件显示所有的图片记录，右击该控件会弹出快捷菜单，选择“获取图片”命令可以从服务器获取图片记录，图片记录显示后在“共享”列中可以设置哪些图片共享，哪些不共享，选择“设置共享”命令会把共享图片的信息发送给服务器，如图2-16所示。



图2-16　设置图片共享

2.4.2 技术要点

1. 共享字段绑定

在 images 数据库表中的 iShared 字段标志该图片是否共享，共享的图片可以被别的用户查看到，不共享的图片别的用户查看不到。iShared 是 bit 类型，值为 1 或者 0，转为 DataTable 后自动转为值为 True 或者 False 的 Bool 类型。这样的字段通过 DataGrid 控件的 CheckBox 列就可以显示为 CheckBox 控件，绑定的方法是：

```
<DataGridCheckBoxColumn Binding="{Binding iShared, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}"
    IsThreeState="False" IsReadOnly="False" Header="共享" Width="50"  />
```

其中绑定中要设置 Mode＝TwoWay 及 UpdateSourceTrigger＝PropertyChanged，否则在用户单击一条记录的 CheckBox 后选择的状态不能及时反映到绑定的数据源对象中。

2. 共享图片的上传

如果客户端用户选择一批要共享的图片记录，那么怎样把这些记录的 ID 值传递给服务器呢？一个方法就是构造一个 List<int>的列表对象，把共享图片的所有 ID 值存储到这个列表中，然后把这个列表序列化成 XML 字符串，再通过 WebClient 的 UploadString 函数把这个字符串发送给服务器，服务器接收后再把该字符串转为 List<int>对象，就可以知道是哪些图片要共享了。

如果客户端与服务器约定 opt＝"setSharedList"为共享图片操作，那么客户端主要代码如下：

```
List<int> IDS=new List<int>();
foreach (DataRow row in dt.Rows)
    if((bool)row["iShared"]) IDS.Add((int)row["ID"]);
Uri uri=new Uri(url+"?opt=setSharedList", UriKind.Absolute);
client.UploadStringAsync(uri, "POST", serialize<List<int>>(IDS));
```

客户端通过 WebClient 的 UploadStringAsync 函数上传要共享的图片 ID 号的列表。

3. 共享图片的设置

服务器端接收到这个 XML 字符串 sXml 后，把它转为 List<int>对象，就可以循环得到共享图片的 ID。修改 DAL 中的 ImageService，设计 getUserImageList 与 setSharedList 函数如下：

```
namespace DAL
{
    public class ImageService
    {
        public String getUserImageList(String uName,String uPass)
```

```
{
    //获取 uName 用户的图片列表
    DataTable dt=new DataTable("images");
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyUser(DB,uName, uPass))
        {
            SqlParameter pName=new SqlParameter("@uName", SqlDbType.
            Char); pName.Value=uName;
            SqlDataAdapter adapter=DB.getAdapter("select ID,uName,
            iDate,iFile,iShared from images where uName=@uName order
            by ID", pName);
            adapter.Fill(dt);
        }
        con.Close();
    }
    return My.serialize<DataTable>(dt);
}
public String setSharedList(String uName, String uPass,String sXml)
{
    String s="failed";
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyUser(DB,uName, uPass))
        {
            //清除共享标志
            SqlParameter pName=new SqlParameter("@uName", SqlDbType.
            Char); pName.Value=uName;
            DB.executeCommand("update images set iShared=0 where uName
            =@uName", pName);
            List<int>IDS=My.deserialize<List<int>>(sXml);
            foreach (int ID in IDS)
            {
                //设置共享标志
                 DB.executeCommand("update images set iShared=1 where
                 ID="+ID.ToString());
            }
            s="shared";
        }
        con.Close();
    }
```

```
            return s;
        }
    }
}
```

其中 getUserImageList 函数是获取图片记录数据集的 XML 序列化结果，setSharedList 函数是把要共享的图片的 iShared 字段设置为 1，不共享的设置为 0。

2.4.3 服务器程序

服务器程序主要由获取用户的图片列表函数 getUserImageList 与设置共享函数 setSharedList 组成，程序如下：

```
…//省略部分 using 代码
using Model;
using MyDLL;
using DAL;
public class ImageServer: IHttpHandler
{
    HttpContext context;
    String getUserImageList()
    {
        //获取用户图片记录
        String uName=My.decode(context.Request.QueryString["uName"]);
        String uPass=My.decode(context.Request.QueryString["uPass"]);
        ImageService imageService=new ImageService();
        return imageService.getUserImageList(uName,uPass);
    }
    String setSharedList()
    {
        //设置共享图片记录
        String uName=My.decode(context.Request.QueryString["uName"]);
        String uPass=My.decode(context.Request.QueryString["uPass"]);
        byte[] buf=context.Request.BinaryRead(context.Request.TotalBytes);
        String sXml=Encoding.UTF8.GetString(buf);
        ImageService imageService=new ImageService();
        return imageService.setSharedList(uName,uPass, sXml);
    }
    public void ProcessRequest(HttpContext context)
    {
        this.context=context;
        ResultClass res=new ResultClass();
        //获取 opt 参数
        String opt=context.Request["opt"];
        try
```

```
        {
            if(opt=="getUserImageList") res.message=getUserImageList();
            else if(opt=="setSharedList") res.message=setSharedList();
            res.state="success";
        }
        catch(Exception exp)
        {
            res.state="error"; res.message=exp.Message;
        }
        context.Response.ContentType="text/plain";
        context.Response.Write(My.serialize<ResultClass>(res));
        context.Response.Flush();
    }
      public bool IsReusable {
        get { return false; }
    }
}
```

2.4.4 客户端程序

1. 界面设计

在客户端的 DataGrid 中主要设计一个弹出式菜单，包含“获取图片”与“设置共享”两个菜单项目。注意绑定列 DataGridCheckBoxColumn 中要设置：

```
Mode=TwoWay, UpdateSourceTrigger=PropertyChanged
```

只有这样才能保证用户在 DataGrid 的 CheckBox 中选择后该信息即刻反映到基础数据源 DataTable 对象中。

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="30" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <StackPanel Orientation="Horizontal">
        <TextBlock Text="用户" VerticalAlignment="Center" />
        <TextBox x:Name="txtUser" Text="xxx" Width="69" VerticalAlignment=
        "Center" Margin="0,6,0,6.4" />
        <TextBlock Text="密码" VerticalAlignment="Center" />
        <PasswordBox x:Name="txtPass" Password="123" Width="69"
        VerticalAlignment="Center" Margin="0,6,0,6.4" />
    </StackPanel>
    <DataGrid x:Name="uGrid" Grid.Row="1" AutoGenerateColumns="False"
    CanUserReorderColumns="False" CanUserResizeRows="False" CanUserAddRows
```

```
        ="False"  SelectionMode="Single">
            <DataGrid.Columns>
                <DataGridTextColumn x:Name="colDate" Binding="{Binding iDate}"
                Header="日期" Width="50" IsReadOnly="True" />
                <DataGridTextColumn x:Name="colFile" Binding="{Binding iFile}"
                Header="文件" Width="50" IsReadOnly="True" />
                <DataGridCheckBoxColumn x:Name="colShared" Binding="{Binding
                iShared, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
                    IsThreeState="False"  IsReadOnly="False" Header="共享" Width
                    ="50"  />
            </DataGrid.Columns>
            <DataGrid.ContextMenu>
                <ContextMenu>
                    <MenuItem x:Name="menuGetImageList" Header="获取图片" Click=
                    "menuGetImageList_Click" />
                    <MenuItem x:Name="menuSetSharedList" Header="设置共享" Click
                    ="menuSetSharedList_Click" />
                </ContextMenu>
            </DataGrid.ContextMenu>
        </DataGrid>
    </Grid>
```

2. 程序设计

客户端程序主要是设置共享的函数中上传的是各个共享图片的 ID 号的列表 List<int>对象,这个对象要序列化后才上传。

```
namespace ImageClient
{
    public partial class MainWindow : Window
    {
        WebClient client;
        String url="http://localhost/web/ImageServer.ashx";
        DataTable dt;
        DataView dv;
        public MainWindow()
        {
            InitializeComponent();
            client=new WebClient();
            client.Encoding=Encoding.UTF8;
            //设置异步回调函数
            client.UploadStringCompleted+=client_UploadStringCompleted;
            client.DownloadStringCompleted+=client_DownloadStringCompleted;
            menuSetSharedList.IsEnabled=false;
```

```
}
String encryptString(String s)
{
    MD5 md5=new MD5CryptoServiceProvider();
    byte[] buf=Encoding.UTF8.GetBytes(s);
    buf=md5.ComputeHash(buf);
    s="";
    foreach (byte x in buf) s=s+x.ToString("X2");
    return s;
}
    void client_DownloadStringCompleted(object sender,
    DownloadStringCompletedEventArgs e)
{
    if(e.Error==null)
    {
        ResultClass res=My.deserialize<ResultClass>(e.Result);
        if(res.state=="success")
        {
            //反序列化成图片数据表
            dt=My.deserialize<DataTable>(res.message);
            //增加一个 iData 字段
            dv=dt.DefaultView;
            dv.Sort="iDate desc";
            uGrid.ItemsSource=dv;
            menuSetSharedList.IsEnabled=true;
        }
        else showMsg(res.message);
    }
    else showMsg(e.Error.Message);
}
void client_UploadStringCompleted(object sender,
UploadStringCompletedEventArgs e)
{
    if(e.Error==null)
    {
        ResultClass res=My.deserialize<ResultClass>(e.Result);
        if(res.state=="success")
        {
            showMsg(res.message);
        }
        else showMsg("设置共享错误!");
    }
    else showMsg(e.Error.Message);
```

```
}
void showMsg(String s)
{
    MessageBox.Show(s, "Information", MessageBoxButton.OK);
}
private void menuGetImageList_Click(object sender, RoutedEventArgs e)
{
    String uName=txtUser.Text.Trim();
    String uPass=txtPass.Password.Trim();
    if(uName !="" && uPass !="")
    {
        uPass=encryptString(uPass);
        try
        {
            Uri uri=new Uri (url+"?opt=getUserImageList&uName="+My.
            encode (uName) +" &uPass =" + My. encode (uPass), UriKind.
            Absolute);
            client.DownloadStringAsync(uri);
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
}
private void menuSetSharedList_Click(object sender, RoutedEventArgs e)
{
    String uName=txtUser.Text.Trim();
    String uPass=txtPass.Password.Trim();
    if(uName !="" && uPass !="")
    {
        uPass=encryptString(uPass);
        try
        {
            //组织共享图片的 ID,组成一个 XML 字符串
            List<int>IDS=new List<int>();
            foreach (DataRow row in dt.Rows)
                if((bool)row["iShared"]) IDS.Add((int)row["ID"]);
            Uri uri=new Uri (url +"? opt = setSharedList&uName =" + My.
            encode (uName) +" &uPass =" + My. encode (uPass), UriKind.
            Absolute);
            client.UploadStringAsync(uri, "POST", My.serialize<List
            <int>>(IDS));
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
```

```
        }
    }
}
```

2.4.5 拓展训练

这个程序中,在执行完设置图片共享后,图片的记录并没有及时更新,如果要及时更新,那么在服务器更新了数据库表 images 后把设置了共享的图片记录的 ID 号再次发送回客户端,客户端及时更新记录。

1. 修改服务器 setSharedList 函数

该函数返回共享图片的 ID 序列 List<int>的 XML 字符串:

```
public String setSharedList(String uName, String uPass, String sXml)
{
    String s="";
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyUser(DB, uName, uPass))
        {
            //清除共享标志
            SqlParameter pName=new SqlParameter("@uName", SqlDbType.Char);
            pName.Value=uName;
            DB.executeCommand("update images set iShared=0 where uName=
            @uName", pName);
            List<int>IDS=My.deserialize<List<int>>(sXml);
            List<int>IDT=new List<int>();
            foreach (int ID in IDS)
            {
                //设置共享标志
                if(DB.executeCommand("update images set iShared=1 where ID="
                +ID.ToString())>0) IDT.Add(ID);
            }
            s=My.serialize<List<int>>(IDT);
        }
        con.Close();
    }
    return s;
}
```

2. 修改客户端 client_UploadStringCompleted 函数

该函数解析服务器传来的共享图片的 ID 序列,把它们设置到数据源中,从而更新显示。

```
void client_UploadStringCompleted(object sender,
UploadStringCompletedEventArgs e)
{
    try
    {
        ResultClass res=My.deserialize<ResultClass>(e.Result);
        if(res.state=="success")
        {
            if(res.message !="")
            {
                //清除共享标志
                foreach (DataRow row in dt.Rows) row["iShared"]=false;
                List<int>IDS=My.deserialize<List<int>>(res.message);
                foreach (int ID in IDS)
                {
                    //设置共享标志
                    DataRow row=dt.AsEnumerable().FirstOrDefault(x=>
                    (int)x["ID"]==ID);
                    if(row !=null) row["iShared"]=true;
                }
                //更新对象
                dt.AcceptChanges();
            }
        }
        else showMsg(res.message);
    }
    catch(Exception exp) { showMsg(exp.Message); }
}
```

这样修改后的程序能确保客户端的共享信息与服务器端的一致。

2.5 图片下载与浏览

2.5.1 案例展示

客户端用一个 DataGrid 列表控件显示出服务器所有的图片记录，当选择其中一条记录时就从服务器下载该图片并显示出来，如图 2-17 所示。



图 2-17 图片浏览

2.5.2 技术要点

1. 图片记录

数据库表 images 中存储了所有的图片，服务器读取该表的所有图片信息，组织成一个 DataTable 表，把这个表序列化成 XML 字符串发送给客户端，客户端接收后再反序列化成 DataTable 对象放在一个 DataGrid 控件上显示。值得注意的是，这个 DataTable 可以包含也可以不包含图片的二进制数据 iData，如果包含，那么这个 DataTable 的数据量是很大的，网络传输会有困难。本项目的程序中暂时先不包含 iData 数据，客户端需要时再去服务器获取。客户端与服务器约定 opt="getSharedImageList"时为获取图片列表。

2. 图片浏览

服务器读取该表的所有图片信息传输给客户端程序，客户端程序使用 DataGrid 控件显示这些图片记录。当用户选择其中一条图片记录就触发 DataGrid 的 SelectionChange 事件，在这个事件函数中获取该记录的 ID 号，通过 WebClient 的 DownloadData 函数从服务器下载图片的二进制数据，该函数的原型如下：

```
public byte[] DownloadData(Uri uri)
```

其中主要的客户端代码如下：

```
DataRowView row=dv[uGrid.SelectedIndex];
String ID=row["ID"].ToString().Trim();
Uri uri=new Uri(url+"?opt=downloadSharedImage&ID="+ID, UriKind.Absolute);
client.DownloadDataAsync(uri, uGrid.SelectedIndex);
```

客户端把这个 ID 号传输给服务器，服务器就可以从数据库中获取该图片的二进制数据。客户端维持一个 DataTable 的对象 dt 与对应的 DataView 对象 dv，它们存储了服务器端的图片记录。

改造服务器 DAL 的 ImageService 类，增加获取所有图片记录的函数 getSharedImageList 与下载某个图片的函数 downloadSharedImage：

```
namespace DAL
{
    public class ImageService
    {
        public String getSharedImageList()
        {
            //获取 uName 用户的图片列表
            DataTable dt=new DataTable("images");
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
```

```
                SqlDataAdapter adapter = DB.getAdapter ( " select  ID, uName,
                iDate,iFile,iShared from images where iShared=1 order by ID");
                adapter.Fill(dt);
                con.Close();
            }
            return My.serialize<DataTable>(dt);
        }
        public byte[] downloadSharedImage(String ID)
        {
            byte[] data=null;
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                SqlDataReader reader=DB.getReader("select * from images where
                ID="+ID+" and iShared=1");
                if(reader.Read())
                {
                    //读取图片二进制数据
                    data=(byte[])reader["iData"];
                }
                reader.Close();
                con.Close();
            }
            return data;
        }
    }
}
```

3. 结果返回

在客户端使用 getSharedImageList 时服务器返回 XML 字符串,服务器设置输出的 ContentType 必须对应为文本,即

```
context.Response.ContentType="text/plain"
```

而当客户端使用 downloadSharedImage 要求服务器返回图片的二进制数据,服务器设置输出的 ContentType 也必须对应为二进制数据,即

```
context.Response.ContentType="application/octet-stream"
```

2.5.3 服务器程序

服务器程序包括返回图片列表的 getSharedImageList 函数与返回图片的 downloadSharedImage 函数,主要程序如下:

```
using Model;
using MyDLL;
using DAL;
public class ImageServer: IHttpHandler
{
    HttpContext context;
    byte[] downloadSharedImage()
    {
        String ID=context.Request.QueryString["ID"];
        ImageService imageService=new ImageService();
        return imageService.downloadSharedImage(ID);
    }
    String getSharedImageList()
    {
        ImageService imageService=new ImageService();
        return imageService.getSharedImageList();
    }
    public void ProcessRequest(HttpContext context)
    {
        this.context=context;
        ResultClass res=new ResultClass();
        byte[] data=null;
        String opt=context.Request.QueryString["opt"];
        try
        {
            if(opt=="getSharedImageList")res.message=getSharedImageList();
            else if(opt=="downloadSharedImage")data=downloadSharedImage();
            res.state="success";
        }
        catch(Exception exp)
        {
            res.state="error"; res.message=exp.Message;
        }
        if(opt=="downloadSharedImage")
        {
            if(data !=null)
            {
                context.Response.ContentType="application/octet-stream";
                context.Response.BinaryWrite(data);
            }
        }
        else
        {
            context.Response.ContentType="text/plain";
```

```
            context.Response.Write(My.serialize<ResultClass>(res));
        }
        context.Response.Flush();
    }
    public bool IsReusable {
        get { return false; }
    }
}
```

编写完毕要执行“重新生成解决方案”更新DAL类库。

2.5.4 客户端程序

1. 界面设计

客户端界面主要是使用DataGrid控件显示服务器发送过来的图片列表，设计该控件的各个列与DataTable表的字段绑定就可以自动显示了，主要代码如下：

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="1*" />
    </Grid.ColumnDefinitions>
    <DataGrid x:Name="uGrid" Grid.Column="0" AutoGenerateColumns="False"
    IsReadOnly="True" SelectionMode="Single" SelectionChanged="uGrid_
    SelectionChanged">
        <DataGrid.Columns>
            <DataGridTextColumn Binding="{Binding ID}" Header="ID" Width=
            "50" />
            <DataGridTextColumn Binding="{Binding uName}" Header="用户"
            Width="50" />
            <DataGridTextColumn Binding="{Binding iDate}" Header="日期"
            Width="100" />
            <DataGridTextColumn Binding="{Binding iFile}" Header="图片"
            Width="100" />
        </DataGrid.Columns>
    </DataGrid>
    <GridSplitter Width="3" />
    <Image x:Name="img" Grid.Column="1" />
</Grid>
```

其中，<GridSplitter Width="3" />是DataGrid与图片控件Image的分隔线。注意，在显示DataTable数据时先获取DataTable的视图对象DataView，然后再把DataView对象绑定到DataGrid控件，这样做的好处是，当单击DataGrid的各个列时该列数据会自动排序，排序后保证DataView对象的记录顺序也跟着变化，只有这样才能正确定位一条记

录的图片。

2. 程序设计

客户端程序包括一个 DataTable 对象 dt 以及对应的 DataView 对象 dv，它们是用来存储图片列表记录的，主要程序如下：

```
public partial class MainWindow : Window
{
    WebClient client;
    String url="http://localhost/web/ImageServer.ashx";
    DataTable dt;
    DataView dv;
    public MainWindow()
    {
        InitializeComponent();
        client=new WebClient();
        client.Encoding=Encoding.UTF8;
        //设置异步回调函数
        client.DownloadStringCompleted+=client_DownloadStringCompleted;
        client.DownloadDataCompleted+=client_DownloadDataCompleted;
    }
    void showImage(byte[] data)
    {
        //显示图片
        try
        {
            BitmapImage bm=new BitmapImage();
            bm.BeginInit();
            bm.StreamSource=new MemoryStream(data);
            bm.EndInit();
            img.Source=bm;
        }
        catch(Exception exp) { showMsg(exp.Message); img.Source=null; }
    }
    void client_DownloadDataCompleted(object sender,
    DownloadDataCompletedEventArgs e)
    {
        //下载图片成功，显示图片
        if(e.Result !=null) showImage(e.Result);
        else img.Source=null;
    }
    void showMsg(String s)
    {
        MessageBox.Show(s, "Information", MessageBoxButton.OK);
```

```
}
void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
{
    if(e.Error==null)
    {
        String s=e.UserState.ToString();
        ResultClass res=My.deserialize<ResultClass>(e.Result);
        if(res.state=="success")
        {
            //如果是 getSharedImageList 触发该事件函数就显示图片列表
            if(s=="getSharedImageList")
            {
                //反序列化成图片数据表
                dt=My.deserialize<DataTable>(res.message);
                dv=dt.DefaultView;
                dv.Sort="iDate desc";
                uGrid.ItemsSource=dv;
            }
        }
        else showMsg(res.message);
    }
    else showMsg(e.Error.Message);
}
private void Window_Loaded(object sender, RoutedEventArgs e)
{
  //程序启动时获取图片列表
    try
    {
        Uri uri=new Uri(url+"?opt=getSharedImageList", UriKind.Absolute);
        //向 DownloadString 发送 getSharedImageList 信息
        client.DownloadStringAsync(uri,"getSharedImageList");
    }
    catch(Exception exp) { showMsg(exp.Message); }
}
private void uGrid_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if(uGrid.SelectedIndex>=0)
    {
        try
        {
            //下载图片,向服务器发送图片的 ID
            DataRowView row=dv[uGrid.SelectedIndex];
```

```
                String ID=row["ID"].ToString().Trim();
                Uri uri=new Uri (url +"? opt = downloadSharedImage&ID =" + ID,
                UriKind.Absolute);
                client.DownloadDataAsync(uri);
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
}
```

程序启动时调用 DownloadStringAsync(uri,"getSharedImageList")获取图片的记录数据集,然后绑定到 DataGrid 显示。在选择一条记录时执行 uGrid_SelectionChanged 函数,根据图片的 ID 号使用 client. DownloadDataAsync 下载图片进行显示。

2.5.5 拓展训练

客户端在每次选择一条图片记录时都会去服务器下载对应图片进行显示,如果浏览过其他记录后再次回到该记录,该图片又会重新下载一次,这样会增加网络的负担。实际上在客户端可以做一个缓存的功能,把下载过的图片缓存到内存中或者磁盘中,下次再需要该图片时先到缓存中去取,如果能取到,就直接取出,而不用再到服务器下载,如果缓存中没有,再去服务器下载。

1. 修改 DownloadStringCompleted 函数

在客户端的 DataTable 对象 dt 中增加一个存储图片二进制数据的字段 iData,函数修改为:

```
void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
{
    if(e.Error==null)
    {
        String s=e.UserState.ToString();
        ResultClass res=My.deserialize<ResultClass>(e.Result);
        if(res.state=="success")
        {
            //如果是 getSharedImageList 触发该事件函数就显示图片列表
            if(s=="getSharedImageList")
            {
                //反序列化成图片数据表
                dt=My.deserialize<DataTable>(res.message);
                //增加一个 iData 字段
                dt.Columns.Add("iData", typeof(byte[]));
                dt.AcceptChanges();
```

```
                dv=dt.DefaultView;
                dv.Sort="iDate desc";
                uGrid.ItemsSource=dv;
            }
        }
        else showMsg(res.message);
    }
    else showMsg(e.Error.Message);
}
```

2. 修改 DownloadDataCompleted 函数

客户端下载图片后缓存图片到 iData 字段，其中 index 是记录的序号：

```
void client_DownloadDataCompleted(object sender,
DownloadDataCompletedEventArgs e)
{
    if(e.Result !=null)
    {
        //下载成功
        showImage(e.Result);
        //存储图片到 index 记录的 iData 字段
        int index=(int)e.UserState;
        dv[index]["iData"]=e.Result;
        dt.AcceptChanges();
    }
    else img.Source=null;
}
```

3. 修改 uGrid_SelectionChanged 函数

在选择图片时先到 dv 的 iData 字段去找图片，如果找不到就到服务器下载，如果有就直接取出来显示：

```
private void uGrid_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    int index=uGrid.SelectedIndex;
    if(index>=0)
    {
        try
        {
            if(dv[index]["iData"]==DBNull.Value)
            {
                //缓存中不存在图片，从服务器下载
```

```
                String ID=dv[index]["ID"].ToString().Trim();
                  Uri uri=new Uri(url+"?opt=downloadSharedImage&ID="+ID,
                  UriKind.Absolute);
                client.DownloadDataAsync(uri,index);
            }
            else
            {
                //从缓存中取出图片显示
                showImage((byte[])dv[index]["iData"]);
            }
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
}
```

经过这样改进后,程序的执行效率会提高,但其缺点是,随着浏览的图片越来越多,客户端的数据集占据的内存会越来越大。

2.6 图片共享程序的实现

2.6.1 技术要点

根据前面的讲解可以看到服务器实际上是一个网页程序,客户端是一个 WPF 的窗体程序,它们之间通过约定的 opt 参数执行不同的操作,实现图片的上传、删除、浏览、共享设置等功能。每次操作时客户端都通过 WebClient 对象与服务器进行数据通信,每次通信时都把 opt 参数传递给服务器,同时传递用户名称与密码以及该操作必要的参数,服务器根据 opt 的值来完成对应的操作。为了传递结构化的类数据,所有的数据往来都采用 XML 序列化。

1. 服务器功能

服务器定义了一组函数完成不同的功能,如表 2-2 所示。

表 2-2 服务器功能

操作命令 opt	调 用 函 数	功 能 说 明
loginUser	String loginUser(String uName, String uPass)	用户注册与登录
downloadSharedImage	byte[] downloadSharedImage(String ID)	下载共享图片
downloadUserImage	byte[] downloadUserImage(String uName, String uPass, String ID)	下载用户图片
deleteImage	String deleteImage(String uName, String uPass, String ID)	删除图片

续表

操作命令 opt	调 用 函 数	功 能 说 明
getUserImageList	String getUserImageList(String uName,String uPass)	获取某个用户的所有图片的列表
getSharedImageList	String getSharedImageList()	获取所有共享图片的列表
uploadImage	String uploadImage (String uName, String uPass, String iFile, byte[] data)	上传某个用户的图片
setSharedList	String setSharedList (String uName, String uPass,String sXml)	设置某个用户共享图片的列表
changePassword	String changePassword(String uName,String uPass,String uNewPass)	更改密码

2. 客户端功能

客户端程序通过 WebClient 调用不同的函数执行不同的功能,如表 2-3 所示。

表 2-3 客户端功能

操作命令 opt	WebClient 调用函数	功 能 说 明
loginUser	DownloadStringAsync	用户注册与登录
downloadSharedImage	DownloadDataAsync	下载共享图片
downloadUserImage	DownloadDataAsync	下载用户图片
deleteImage	DownloadStringAsync	删除图片
getUserImageList	DonloadStringAsync	获取某个用户的所有图片的列表
getSharedImageList	DownloadStringAsync	获取所有共享图片的列表
uploadImage	UploadDataAsync	上传某个用户的图片
setSharedList	UploadStringAsync	设置某个用户共享图片
changePassword	DownloadStringAsync	更改密码

3. 客户端信息

客户端由于有多个操作都调用 DownloadStringAsync 函数,因此服务器返回时都触发相同的 DownloadCompleted 函数。为了区别是基于什么功能操作调用的 DownloadStringAsync 函数以便程序执行不同的操作,在 Model 中设计一个信息类 MessageClass:

```
public class MessageClass
{
    public String opt { get; set; }
    public String message { get; set; }
```

```
}
```

其中 opt 表示操作命令参数，message 是附带的信息。这样在调用 DownloadStringAsync 时只要同时向它传递一个 MessageClass 对象，就可以知道是哪里调用的了。例如，客户端要删除服务器的一张图片，就执行以下代码：

```
DataRowView row=dv[uGrid.SelectedIndex];
String ID=row["ID"].ToString().Trim();
Uri uri=new Uri(url+"?opt=deleteImage&ID="+ID, UriKind.Absolute);
client. DownloadStringAsync (uri, new MessageClass { opt =" deleteImage ",
message=uGrid.SelectedIndex.ToString() });
```

在调用 DownloadStringAsync 时传递一个 MessageClass 对象，该对象包含 opt＝"deleteImage"的操作命令，一个选择记录的序号。在服务器返回信息触发 DownloadStringCompleted 函数时就执行以下代码：

```
if(msg.opt=="deleteImage")
{
    //删除图片
    if(res.message=="deleted")
    {
        int index=int.Parse(msg.message);
        dv[index].Delete();
        dt.AcceptChanges();
    }
}
```

通过这样的设置，DownloadStringCompleted 中就知道用户是要删除哪个序号的记录，以便执行删除该记录的操作。

4. 用户登录窗体

本项目另外设计一个登录窗体来接收用户名称与密码的输入，窗体名称为 LoginWindow，在执行登录时创建该窗体对象 dlg，然后用模态的方式显示此窗体，在此窗体类中设计共有的变量 state、uName 及 uPass，窗体关闭后读取这些变量就可以得到用户登录的信息。主要的代码如下：

```
LoginWindow dlg=new LoginWindow();
dlg.Owner=this;
dlg.ShowDialog();
if(dlg.state=="OK")
{
    String uName=dlg.uName;
    String uPass=dlg.uPass;
    …
}
```

同样的道理，另外设计一个窗体 PasswordWindow 用于更改密码。

2.6.2 服务器程序

综合前面各节，服务器程序的结构如图 2-18 所示。

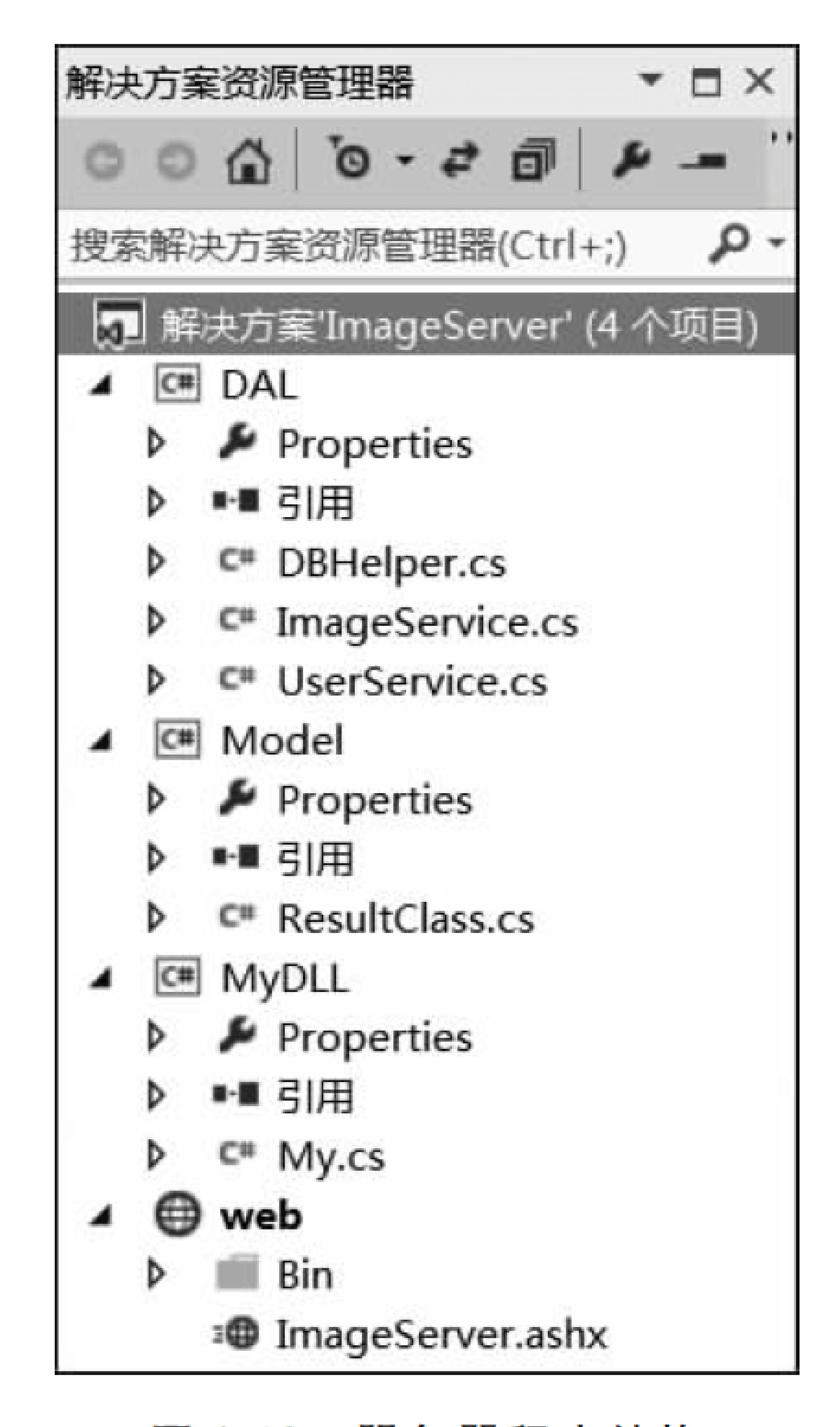


图 2-18 服务器程序结构

1. DAL 中的 UserService

```
namespace DAL
{
    public class UserService
    {
        public String login(String uName, String uPass)
        {
            String s="failed";
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                SqlParameter pName=new SqlParameter("@uName", SqlDbType.
                Char); pName.Value=uName;
                SqlParameter pPass=new SqlParameter("@uPass", SqlDbType.
                Char); pPass.Value=uPass;
                try
                {
```

```
                DB.executeCommand("insert into users values (@uName,
                @uPass)", pName, pPass);
                s="registered";
            }
            catch
            {
                if(DB.getScalar("select count(*) from users where uName=
                @uName and uPass=@uPass", pName, pPass)>0) s="logined"; ;
            }
            con.Close();
        }
        return s;
    }
    public String changePassword(String uName, String uPass,String uNewPass)
    {
        String s="failed";
        using(SqlConnection con=new SqlConnection(DBHelper.conString))
        {
            DBHelper DB=new DBHelper(con);
            SqlParameter pNewPass = new SqlParameter ( "@uNewPass",
            SqlDbType.Char); pNewPass.Value=uNewPass;
            SqlParameter pName = new SqlParameter ("@uName", SqlDbType.
            Char); pName.Value=uName;
            SqlParameter pPass = new SqlParameter ("@uPass", SqlDbType.
            Char); pPass.Value=uPass;
            if(DB.executeCommand("update users set uPass=@uNewPass where
            uName=@uName and uPass=@uPass", pNewPass, pName, pPass)>0) s
            ="changed";
            con.Close();
        }
        return s;
    }
  }
}
```

2. DAL 中的 ImageService

```
namespace DAL
{
    public class ImageService
    {
        bool verifyUser(DBHelper DB,String uName, String uPass)
        {
            SqlParameter pName=new SqlParameter("@uName", SqlDbType.Char);
```

```
    pName.Value=uName;
    SqlParameter pPass=new SqlParameter("@uPass", SqlDbType.Char);
    pPass.Value=uPass;
    return((int)DB.getScalar("select count(*) from users where uName
    =@uName and uPass=@uPass", pName, pPass)>0);
}
public String getSharedImageList()
{
    //获取 uName 用户的图片列表
    DataTable dt=new DataTable("images");
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        SqlDataAdapter adapter=DB.getAdapter("select ID,uName,iDate,
        iFile,iShared from images where iShared=1 order by ID");
        adapter.Fill(dt);
        con.Close();
    }
    return My.serialize<DataTable>(dt);
}
public String getUserImageList(String uName,String uPass)
{
    //获取 uName 用户的图片列表
    DataTable dt=new DataTable("images");
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyUser(DB,uName, uPass))
        {
            SqlParameter pName=new SqlParameter("@uName", SqlDbType.
            Char); pName.Value=uName;
            SqlDataAdapter adapter=DB.getAdapter("select ID,uName,
            iDate,iFile,iShared from images where uName=@uName order
            by ID", pName);
            adapter.Fill(dt);
        }
        con.Close();
    }
    return My.serialize<DataTable>(dt);
}
public String uploadImage(String uName, String uPass, String iFile,
byte[] data)
{
    String ID="0";
```

```
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyUser(DB,uName, uPass))
        {
            SqlParameter pName=new SqlParameter("@uName", SqlDbType.
            Char); pName.Value=uName;
            SqlParameter pDate=new SqlParameter("@iDate", SqlDbType.
            Char); pDate.Value=DateTime.Now.ToString("yyyy-MM-dd HH:
            mm:ss");
            SqlParameter pFile=new SqlParameter("@iFile", SqlDbType.
            Char); pFile.Value=iFile;
            SqlParameter pData=new SqlParameter("@iData", SqlDbType.
            Image); pData.Value=data;
            if(DB.executeCommand("insert into images (uName, iDate,
            iFile,iData) values (@uName, @iDate, @iFile, @iData)",
            pName, pDate, pFile, pData)>0)
            {
                //插入成功时获取插入记录的 ID
                ID=DB.getScalar("select top 1 ID from images order by ID
                desc").ToString();
            }
        }
        con.Close();
    }
    return ID;
}
public byte[] downloadSharedImage(String ID)
{
    byte[] data=null;
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        SqlDataReader reader=DB.getReader("select * from images where
        ID="+ID+" and iShared=1");
        if(reader.Read())
        {
            //读取图片二进制数据
            data=(byte[])reader["iData"];
        }
        reader.Close();
        con.Close();
    }
    return data;
```

```
}
public byte[] downloadUserImage(String uName,String uPass,String ID)
{
    byte[] data=null;
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyUser(DB,uName, uPass))
        {
            SqlDataReader reader= DB.getReader("select * from images
            where ID="+ID);
            if(reader.Read())
            {
                //读取图片二进制数据
                data= (byte[])reader["iData"];
            }
            reader.Close();
        }
        con.Close();
    }
    return data;
}
public String deleteImage(String uName, String uPass, String ID)
{
    String s="failed";
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyUser(DB,uName, uPass))
        {
            if(DB.executeCommand("delete from images where ID="+ID)>
           0) s="deleted";
        }
        con.Close();
    }
    return s;
}
public String setSharedList(String uName, String uPass,String sXml)
{
    String s="failed";
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyUser(DB,uName, uPass))
```

```
                {
                    //清除共享标志
                    SqlParameter pName=new SqlParameter("@uName", SqlDbType.
                    Char); pName.Value=uName;
                    DB.executeCommand("update images set iShared=0 where uName
                    =@uName", pName);
                    List<int>IDS=My.deserialize<List<int>>(sXml);
                    foreach (int ID in IDS)
                    {
                        //设置共享标志
                        DB.executeCommand("update images set iShared=1 where
                        ID="+ID.ToString());
                    }
                    s="shared";
                }
                con.Close();
            }
            return s;
        }
    }
}
```

3. Web 中的 ImageServer. ashx

```
using Model;
using MyDLL;
using DAL;
public class server : IHttpHandler
{
    HttpContext context;
    String loginUser()
    {
        //注册与登录
        String uName=My.decode(context.Request.QueryString["uName"]);
        String uPass=My.decode(context.Request.QueryString["uPass"]);
        UserService userService=new UserService();
        return userService.login(uName, uPass);
    }
    String changePassword()
    {
        //更改密码
        String uName=My.decode(context.Request.QueryString["uName"]);
        String uPass=My.decode(context.Request.QueryString["uPass"]);
        String uNewPass=My.decode(context.Request.QueryString["uNewPass"]);
```

```
        UserService userService=new UserService();
        return userService.changePassword(uName, uPass,uNewPass);
    }
    String uploadImage()
    {
        //上传图片
        String uName=My.decode(context.Request.QueryString["uName"]);
        String uPass=My.decode(context.Request.QueryString["uPass"]);
        String iFile=My.decode(context.Request.QueryString["iFile"]);
        byte[] data=context.Request.BinaryRead(context.Request.TotalBytes);
        ImageService imageService=new ImageService();
        return imageService.uploadImage(uName,uPass,iFile, data);
    }
    byte[] downloadSharedImage()
    {
        //下载共享图片
        String ID=context.Request.QueryString["ID"];
        ImageService imageService=new ImageService();
        return imageService.downloadSharedImage(ID);
    }
    byte[] downloadUserImage()
    {
        //下载用户图片
        String uName=My.decode(context.Request.QueryString["uName"]);
        String uPass=My.decode(context.Request.QueryString["uPass"]);
        String ID=context.Request.QueryString["ID"];
        ImageService imageService=new ImageService();
        return imageService.downloadUserImage(uName,uPass,ID);
    }
    String getSharedImageList()
    {
        //获取共享图片记录
        ImageService imageService=new ImageService();
        return imageService.getSharedImageList();
    }
    String getUserImageList()
    {
        //获取用户图片记录
        String uName=My.decode(context.Request.QueryString["uName"]);
        String uPass=My.decode(context.Request.QueryString["uPass"]);
        ImageService imageService=new ImageService();
        return imageService.getUserImageList(uName,uPass);
    }
    String setSharedList()
```

```
{
    //设置共享图片记录
    String uName=My.decode(context.Request.QueryString["uName"]);
    String uPass=My.decode(context.Request.QueryString["uPass"]);
    byte[] buf=context.Request.BinaryRead(context.Request.TotalBytes);
    String sXml=Encoding.UTF8.GetString(buf);
    ImageService imageService=new ImageService();
    return imageService.setSharedList(uName,uPass, sXml);
}
String deleteImage()
{
    //删除图片
    String uName=My.decode(context.Request.QueryString["uName"]);
    String uPass=My.decode(context.Request.QueryString["uPass"]);
    String ID=context.Request.QueryString["ID"];
    ImageService imageService=new ImageService();
    return imageService.deleteImage(uName, uPass, ID);
}
public void ProcessRequest(HttpContext context)
{
    this.context=context;
    ResultClass res=new ResultClass();
    byte[] data=null;
    //获取 opt 参数
    String opt=context.Request["opt"];
    try
    {
        if(opt=="loginUser") res.message=loginUser();
        else if(opt=="changePassword") res.message=changePassword();
        else if(opt=="getSharedImageList") res.message=getSharedImageList();
        else if(opt=="getUserImageList") res.message=getUserImageList();
        else if(opt=="downloadSharedImage") data=downloadSharedImage();
        else if(opt=="downloadUserImage") data=downloadUserImage();
        else if(opt=="uploadImage") res.message=uploadImage();
        else if(opt=="deleteImage") res.message=deleteImage();
        else if(opt=="setSharedList") res.message=setSharedList();
        res.state="success";
    }
    catch(Exception exp)
    {
        res.state="error"; res.message=exp.Message;
    }
    context.Response.Clear();
    if(opt=="downloadSharedImage"||opt=="downloadUserImage")
```

```
        {
            if(data !=null)
            {
                context.Response.ContentType="application/octet-stream";
                context.Response.BinaryWrite(data);
            }
        }
        else
        {
            context.Response.ContentType="text/plain";
            context.Response.Write(My.serialize<ResultClass>(res));
        }
        context.Response.Flush();
    }
    public bool IsReusable {
        get { return false; }
    }
}
```

2.6.3 客户端程序

综合前面的各节,客户端程序结构如图 2-19 所示。

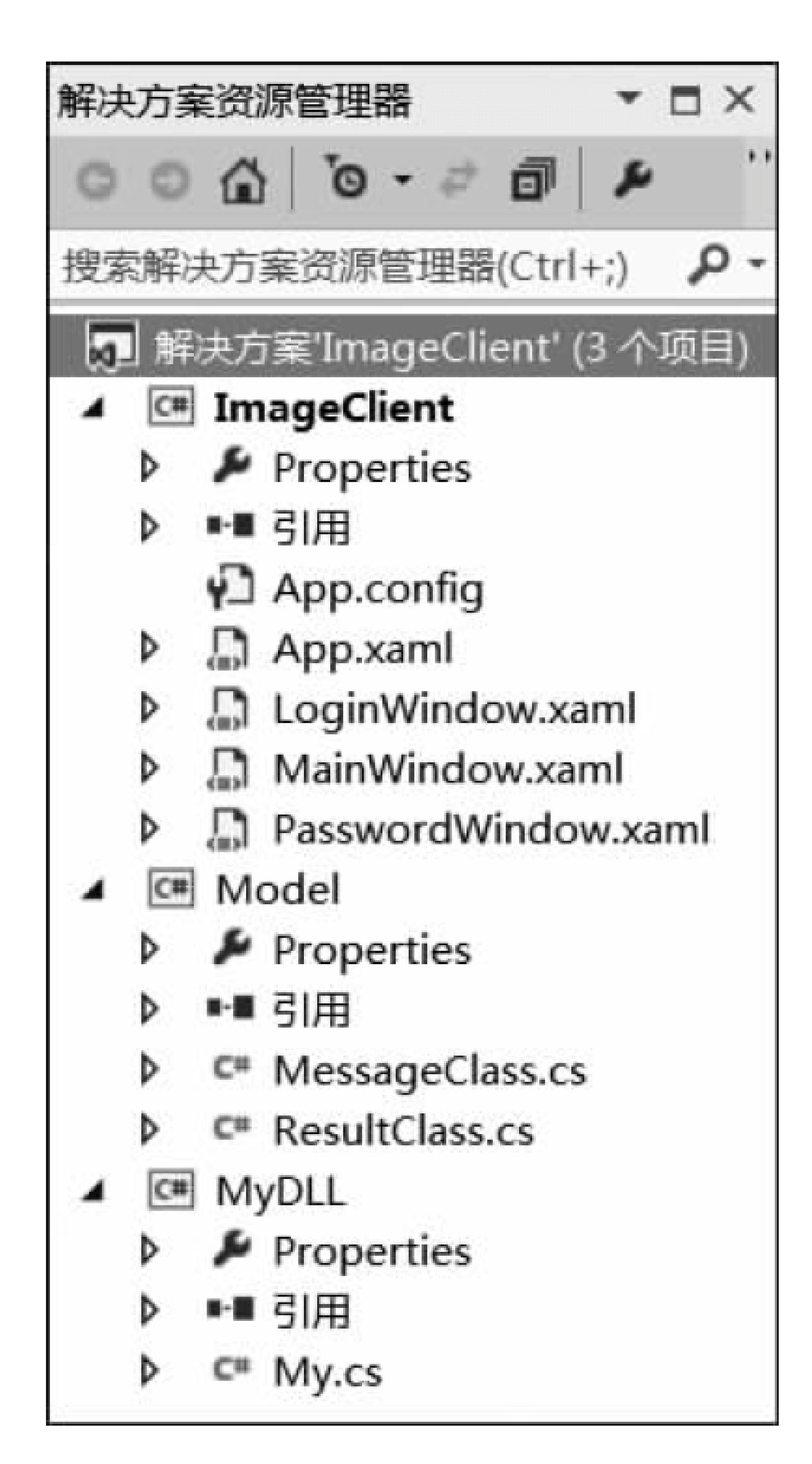


图 2-19 客户端程序结构

1. 界面设计

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="30" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid Grid.Row="0">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="100" />
        </Grid.ColumnDefinitions>
        <TextBox x:Name="txtUrl" VerticalAlignment="Center" Text="http://
        localhost/web/ImageServer.ashx" Grid.Column="0" />
        <Button x:Name="btCon" Content="连接" VerticalAlignment="Center"
        Height="25" Width="50" Grid.Column="1" Click="btCon_Click" />
    </Grid>
    <Grid Grid.Row="1">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="200" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>
        <DataGrid x:Name="uGrid"  AutoGenerateColumns="False"
        SelectionMode="Single"
        SelectionChanged="uGrid_SelectionChanged"
        CanUserResizeRows="False" CanUserAddRows="False" Grid.Column="0"
        CanUserReorderColumns="False">
            <DataGrid.Columns>
                <DataGridTextColumn x: Name =" colName "  Binding =" {Binding
                uName}" Header="用户" Width="100" IsReadOnly="True" />
                <DataGridTextColumn x: Name =" colDate "  Binding =" {Binding
                iDate}" Header="日期" Width="100" IsReadOnly="True" />
                <DataGridTextColumn x: Name =" colFile "  Binding =" {Binding
                iFile}" Header="文件" Width="100" IsReadOnly="True" />
                <DataGridCheckBoxColumn x:Name="colShared" Binding="{Binding
                iShared, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
                IsThreeState="False" IsReadOnly="False" Header="共享" Width=
                "100" Visibility="Hidden" />
            </DataGrid.Columns>
            <DataGrid.ContextMenu>
                <ContextMenu x:Name="gridMenu" Opened="gridMenu_Opened">
                    <MenuItem x:Name="menuLogin" Header="用户登录"
                    Click="menuLogin_Click"  />
                    <MenuItem x:Name="menuChangePassword" Header="更改密码"
```

```
                    Click="menuChangePassword_Click"  />
                    <Separator/>
                    <MenuItem x:Name="menuUploadImage" Header="上传图片"
                    Click="menuUploadImage_Click" />
                    <MenuItem x:Name="menuDeleteImage" Header="删除图片"
                    Click="menuDeleteImage_Click" />
                    <MenuItem x:Name="menuSetSharedImage" Header="设置共享"
                    Click="menuSetSharedImage_Click" />
                    <Separator/>
                    <MenuItem x:Name="menuSaveAsImage" Header="另存图片"
                    Click="menuSaveAsImage_Click" />
                    <Separator/>
                    <MenuItem x:Name="menuRefreshImage" Header="刷新图片"
                    Click="menuRefreshImage_Click" />
                </ContextMenu>
            </DataGrid.ContextMenu>
        </DataGrid>
        <GridSplitter Width="3" Background="Red" />
        <Image x:Name="img" Stretch="Fill" Grid.Column="1"/>
    </Grid>
</Grid>
```

2. 程序设计

```
namespace ImageClient
{
    public partial class MainWindow : Window
    {
        WebClient client;
        String url="http://localhost/web/ImageServer.ashx";
        //当前用户
        //currentName!=""时表示有用户登录
        String currentName="",currentPass="";
        //图片记录表与视图
        DataTable dt;
        DataView dv;
        public MainWindow()
        {
            InitializeComponent();
            client=new WebClient();
            client.Encoding=Encoding.UTF8;
            //设置异步函数
            client.DownloadStringCompleted+=client_DownloadStringCompleted;
            client.UploadDataCompleted+=client_UploadDataCompleted;
```

```
    client.DownloadDataCompleted+=client_DownloadDataCompleted;
    client.UploadStringCompleted+=client_UploadStringCompleted;
}
void showMsg(String s)
{
    MessageBox.Show(s, "Information", MessageBoxButton.OK);
}
bool confirm(String s)
{
    return(MessageBox.Show(s, "Information", MessageBoxButton.YesNo)
    ==MessageBoxResult.Yes);
}
String encryptString(String s)
{
    MD5 md5=new MD5CryptoServiceProvider();
    byte[] buf=Encoding.UTF8.GetBytes(s);
    buf=md5.ComputeHash(buf);
    s="";
    foreach (byte x in buf) s=s+x.ToString("X2");
    return s;
}
void client_UploadStringCompleted(object sender,
UploadStringCompletedEventArgs e)
{
    //设置图片共享后触发此函数
    try
    {
        ResultClass res=My.deserialize<ResultClass>(e.Result);
        if(res.state=="success")
        {
            if(res.message=="shared") showMsg("共享图片成功!");
            else showMsg("共享图片失败!");
        }
        else showMsg(res.message);
    }
    catch(Exception exp) { showMsg(exp.Message); }
}
void showImage(byte[] data)
{
    //显示图片
    try
    {
        BitmapImage bm=new BitmapImage();
        bm.BeginInit();
```

```
            bm.StreamSource=new MemoryStream(data);
            bm.EndInit();
            img.Source=bm;
        }
        catch(Exception exp) { showMsg(exp.Message); img.Source=null; }
    }
    void client_DownloadDataCompleted(object sender,
    DownloadDataCompletedEventArgs e)
    {
         //下载图片成功后触发此函数
        if(e.Result !=null) showImage(e.Result);
        else img.Source=null;
    }
    void client_UploadDataCompleted(object sender,
    UploadDataCompletedEventArgs e)
    {
        //上传图片完成后触发此函数
        try
        {
            String s=Encoding.UTF8.GetString(e.Result);
            ResultClass res=My.deserialize<ResultClass>(s);
            if(res.state=="success")
            {
                if(res.message !="0")
                {
                    DataRow row=dt.NewRow();
                    row["ID"]=res.message;
                    row["uName"]=currentName;
                    row["iFile"]=e.UserState.ToString();
                    row["iDate"]=DateTime.Now.ToString("yyyy-MM-dd
                    HH:mm:ss");
                    row["iShared"]=false;
                    dt.Rows.Add(row);
                    dt.AcceptChanges();
                }
                else showMsg("图片上传失败!");
            }
            else showMsg(res.message);
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
    void client_DownloadStringCompleted(object sender,
    DownloadStringCompletedEventArgs e)
    {
```

```
//获取图片记录、用户注册登录、删除图片、更改密码触发此函数
try
{
    MessageClass msg= (MessageClass)e.UserState;
    ResultClass res=My.deserialize<ResultClass>(e.Result);
    if(res.state=="success")
    {
        if(msg.opt=="getSharedImageList")
        {
            //获取共享图片记录表
            dt=My.deserialize<DataTable>(res.message);
            dt.Columns.Add("iData", typeof(byte[]));
            dv=dt.DefaultView;
            dv.Sort="iDate desc";
            uGrid.ItemsSource=dv;
            colShared.Visibility=Visibility.Hidden;
            if(msg.message=="connect")
            {
                //如果是连接则设置
                txtUrl.IsEnabled=false;
                btCon.Content="断开";
            }
        }
        else if(msg.opt=="loginUser")
        {
            //用户注册或者登录后重新获取用户所拥有的图片记录表
            if(res.message=="logined" || res.message=="registered")
            {
                currentName=msg.message;
                //currentPass 已经设置
                Uri uri=new Uri(url+"?opt=getUserImageList&uName
                ="+My.encode(currentName)+"&uPass="+My.encode
                (currentPass), UriKind.Absolute);
                client.DownloadStringAsync(uri, new MessageClass
                { opt="getUserImageList", message=currentName });
            }
            else
            {
                showMsg("用户登录或者注册失败");
                currentName="";
                currentPass="";
            }
        }
        else if(msg.opt=="getUserImageList")
```

```
                {
                    //获取用户所拥有的图片记录表
                    dt=My.deserialize<DataTable>(res.message);
                    dt.Columns.Add("iData", typeof(byte[]));
                    dv=dt.DefaultView;
                    dv.Sort="iDate desc";
                    uGrid.ItemsSource=dv;
                    colShared.Visibility=Visibility.Visible;
                }
                else if(msg.opt=="deleteImage")
                {
                    //删除图片
                    if(res.message=="deleted")
                    {
                        int index=int.Parse(msg.message);
                        dv[index].Delete();
                        dt.AcceptChanges();
                    }
                }
                else if(msg.opt=="changePassword")
                {
                    //更改密码
                    currentPass=msg.message;
                    showMsg(res.message);
                }
            }
            else showMsg(res.message);
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
    private void uGrid_SelectionChanged(object sender,
    SelectionChangedEventArgs e)
    {
        int index=uGrid.SelectedIndex;
        if( index>=0)
        {
            try
            {
                if(dv[index]["iData"]==DBNull.Value)
                {
                    //如果图片不在缓存中就去服务器下载
                    DataRowView row=dv[index];
                    String ID=row["ID"].ToString().Trim();
                    if(currentName=="")
```

```
                    {
                        Uri uri=new Uri(url+"?opt=downloadSharedImage&ID="
                        +ID, UriKind.Absolute);
                        client.DownloadDataAsync(uri, index);
                    }
                    else
                    {
                        Uri uri=new Uri(url+"?opt=downloadUserImage&ID="
                        +ID+"&uName="+My.encode(currentName)+"&uPass="+
                       My.encode(currentPass), UriKind.Absolute);
                        client.DownloadDataAsync(uri, index);
                    }
                }
                else
                {
                    //如果图片在缓存中就直接显示
                    byte[] data=(byte[])dv[index]["iData"];
                    showImage(data);
                }
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
    private void btCon_Click(object sender, RoutedEventArgs e)
    {
        if(btCon.Content.ToString()=="连接")
        {
            url=txtUrl.Text.Trim();
            if(url !="")
            {
                try
                {
                    //连接成功后获取共享图片记录表
                    Uri uri=new Uri(url+"?opt=getSharedImageList",
                    UriKind.Absolute);
                    client.DownloadStringAsync(uri, new MessageClass { opt
                    ="getSharedImageList", message="connect" });
                }
                catch(Exception exp) { showMsg(exp.Message); }
            }
        }
        else if(confirm("确实要断开连接吗?"))
        {
            //断开连接后删除所有显示
```

```
                btCon.Content="连接";
                txtUrl.IsEnabled=true;
                dt=null;
                dv=null;
                uGrid.ItemsSource=null;
                img.Source=null;
                menuLogin.IsEnabled=false;
                currentName="";
                currentPass="";
            }
        }
        private void menuLogin_Click(object sender, RoutedEventArgs e)
        {
            if(currentName=="")
            {
                //用户注册登录
                LoginWindow dlg=new LoginWindow();
                dlg.Owner=this;
                dlg.ShowDialog();
                if(dlg.state=="OK")
                {
                    String uName=dlg.uName;
                    String uPass=dlg.uPass;
                    if(uName !="" && uPass !="")
                    {
                        //注册登录成功后获取用户所拥有的所有图片记录表
                        try
                        {
                            //设置 currentPass
                            currentPass=encryptString(uPass);
                            Uri uri=new Uri(url+"?opt=loginUser&uName="+My.
                            encode(uName)+"&uPass="+My.encode(currentPass),
                            UriKind.Absolute);
                            client.DownloadStringAsync(uri, new MessageClass
                            { opt="loginUser", message=uName });
                        }
                        catch(Exception exp) { showMsg(exp.Message); }
                    }
                }
            }
            else
            {
                //用户注销,获取所有共享图片记录表
                currentName="";
```

```
            currentPass="";
            try
            {
                Uri uri=new Uri(url+"?opt=getSharedImageList",
                UriKind.Absolute);
                client.DownloadStringAsync(uri, new MessageClass { opt =
                "getSharedImageList", message="" });
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
    private void menuUploadImage_Click(object sender, RoutedEventArgs e)
    {
        OpenFileDialog dlg=new OpenFileDialog();
        dlg.Filter="Images|*.jpg;*.png";
        if((bool)dlg.ShowDialog())
        {
            //上传图片
            try
            {
                String fn=dlg.FileName;
                int p=fn.LastIndexOf("\\");
                if(p>=0) fn=fn.Substring(p+1);
                DataRow row=dt.AsEnumerable().FirstOrDefault(x=>
                x["iFile"].ToString().Trim().ToLower()==fn.ToLower());
                if(row==null)
                {
                    FileStream fs=new FileStream(dlg.FileName,
                    FileMode.Open);
                    byte[] data=new byte[fs.Length];
                    fs.Read(data, 0, data.Length);
                    fs.Close();
                    Uri uri=new Uri(url+"?opt=uploadImage&uName="+
                    My.encode(currentName)+"&uPass="+
                    My.encode(currentPass)+"&iFile="+My.encode(fn),
                    UriKind.Absolute);
                    client.UploadDataAsync(uri, "POST", data, fn);
                }
                else showMsg(fn+"已经存在!");
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
    private void menuSetSharedImage_Click(object sender, RoutedEventArgs e)
```

```
{
    if(currentName !="")
    {
        try
        {
            //组织共享图片的 ID,组成一个 XML 字符串
            List<int>IDS=new List<int>();
            foreach (DataRow row in dt.Rows)
                if((bool)row["iShared"]) IDS.Add((int)row["ID"]);
            Uri uri=new Uri(url+"?opt=setSharedList&uName="+
            My.encode(currentName)+"&uPass="+
            My.encode(currentPass), UriKind.Absolute);
            client.UploadStringAsync(uri, "POST", My.serialize<List
            <int>>(IDS));
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
}
private void menuDeleteImage_Click(object sender, RoutedEventArgs e)
{
    if(uGrid.SelectedIndex>=0)
    {
        if(confirm("确实要删除该图片吗?"))
        {
            try
            {
                DataRowView row=dv[uGrid.SelectedIndex];
                String ID=row["ID"].ToString().Trim();
                Uri uri=new Uri(url+"?opt=deleteImage&ID="+ID+
                "&uName="+My.encode(currentName)+"&uPass="+
                My.encode(currentPass), UriKind.Absolute);
                client.DownloadStringAsync(uri, new MessageClass { opt
                ="deleteImage", message=uGrid.SelectedIndex.ToString
                () });
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
}
private void menuRefreshImage_Click(object sender, RoutedEventArgs e)
{
    //刷新图片记录
    try
    {
```

```
            Uri uri=new Uri(url+"?opt=getSharedImageList",
            UriKind.Absolute);
            client.DownloadStringAsync(uri, new MessageClass { opt=
            "getSharedImageList", message="" });
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
    private void menuChangePassword_Click(object sender, RoutedEventArgs e)
    {
        //修改密码
        PasswordWindow dlg=new PasswordWindow();
        dlg.ShowDialog();
        if(dlg.res=="OK")
        {
            String uNewPass=encryptString(dlg.uPass);
            try
            {
                Uri uri=new Uri(url+"?opt=changePassword&uName="+
                My.encode(currentName)+"&uPass="+My.encode(currentPass)+
               "&uNewPass="+My.encode(uNewPass) , UriKind.Absolute);
                client.DownloadStringAsync(uri, new MessageClass { opt=
                "changePassword", message=uNewPass });
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
    private void menuSaveAsImage_Click(object sender, RoutedEventArgs e)
    {
        //图片另存到本地磁盘
        int index=uGrid.SelectedIndex;
        if(index>=0)
        {
            try
            {
                String fileName=dv[index]["iFile"].ToString();
                SaveFileDialog dlg=new SaveFileDialog();
                dlg.FileName=fileName;
                if((bool)dlg.ShowDialog())
                {
                    FileStream fs=new FileStream(dlg.FileName,
                    FileMode.Create);
                    byte[] data=(byte[])dv[index]["iData"];
                    fs.Write(data, 0, data.Length);
```

```
                        fs.Close();
                    }
                }
                catch(Exception exp) { showMsg(exp.Message); }
            }
        }
        private void gridMenu_Opened(object sender, RoutedEventArgs e)
        {
            //弹出菜单时确定哪些菜单是可以使用的,哪些是不可以使用的
            if(currentName=="")
            {
                menuLogin.Header="用户登录";
                menuChangePassword.IsEnabled=false;
                menuDeleteImage.IsEnabled=false;
                menuUploadImage.IsEnabled=false;
                menuSetSharedImage.IsEnabled=false;
                menuRefreshImage.IsEnabled=(btCon.Content.ToString()=="断开");
            }
            else
            {
                menuLogin.Header="用户注销";
                menuChangePassword.IsEnabled=true;
                menuDeleteImage.IsEnabled=(uGrid.SelectedIndex>=0);
                menuUploadImage.IsEnabled=true;
                menuSetSharedImage.IsEnabled=true;
                menuRefreshImage.IsEnabled=false;
            }
            menuLogin.IsEnabled=(btCon.Content.ToString()=="断开");
            menuSaveAsImage.IsEnabled=(uGrid.SelectedIndex>=0);
        }
    }
}
```

2.6.4 拓展训练

这个项目中服务器的功能是维护管理一组用户与图片,没有任何界面,客户端采用WPF设计成窗体界面显示用户与图片信息。实际上这样的服务器是通用的,也可以设计其他客户端来与该服务器进行交互。例如把客户端设计成一个网页界面,通过浏览器来浏览,或者把客户端设计成一个Android或者iOS的手机App等都是可以的,有兴趣的读者可以尝试。

练　习　二

1. 如果把服务器做成一个普通网页 server.aspx，客户端是一个 WPF 程序，编写服务器与客户端程序，客户端向服务器发送用户名称 uName 与密码 uPass 信息，服务器获取后把它们发回客户端，比较网页服务器 server.aspx 与一般服务器程序 server.ashx 的差异。

2. 定义一个用户信息类 UserClass：

```
public class UserClass
{
    public String uName { get; set; }
    public String uPass { get; set; }
}
```

客户端程序把用户信息用 XML 序列化后使用 WebClient 的 UploadString 把字符串提交给服务器程序，服务器接收后反序列化成 UserClass 对象。

3. 定义一个文件类 FileClass：

```
public class FileClass
{
    public String Name { get; set; }
    public byte[] Data { get; set; }
}
```

其中 Name、Data 分别是文件名称与文件数据，编写客户端与服务器程序，客户端确定一个上传的文件后生成一个文件对象，把这个文件对象序列化成 XML 字符串上传给服务器，服务器接收后反序列化成文件对象。

4. 客户端上传一个二进制文件给服务器时必须上传文件名称与文件数据，如果规定只能使用 WebClient 的 UploadData 一次上传完毕，那么可以考虑把文件名称转为二进制数据与文件数据一起上传，可以这样组合二进制数据：

第一部分，用开始的两个字节记录文件名称的二进制数据长度。

第二部分，文件名称的二进制数据。

第三部分，文件的二进制数据。

编写客户端程序与服务器程序，客户端确定一个上传文件，并把文件数据按这个要求组合，然后用 UploadData 函数一次性上传给服务器，服务器接收这组二进制数据后分解出文件名称与实际的文件数据，把文件存储到服务器磁盘中。

项目三



基于微信的成绩查询程序

Web Service 是类似于网页的一种 Web 服务，但又不同于网页，网页提供界面与数据供用户浏览，Web Service 只提供函数供客户端的程序调用，客户端通过一定的协议调用服务器的 Web Service 的函数实现与服务器的通信。

微信是目前最常用的 App 之一，基于微信的应用越来越多，它使得移动端的程序不再依赖专用的 App，而是使用微信这个特殊的 App 就能完成移动开发。

微信成绩查询系统后台基于 Web Service 实现成绩管理，前台基于微信查询成绩。成绩管理部分包括客户端与服务器两大部分，客户端程序登录后就可以实现课程记录管理、学生记录管理、成绩记录管理。

图 3-1 所示为程序结构，手机用户通过微信服务器与成绩服务器进行交互，计算机客户端直接与成绩服务器进行交互。

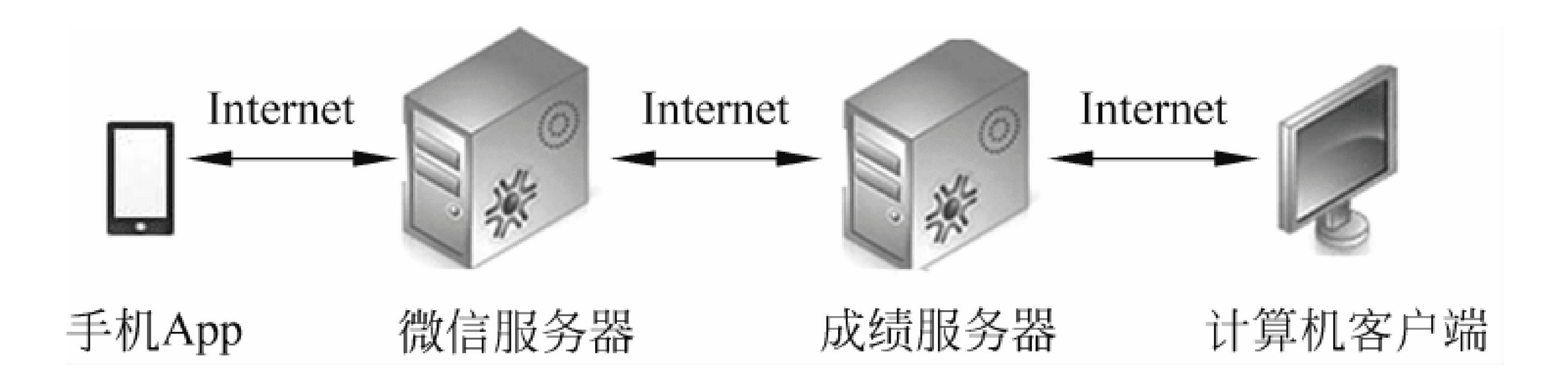


图 3-1　程序结构

图 3-2 至图 3-5 为客户端程序的各个功能界面，管理员可以对课程、学生以及每个学生每门课程的成绩进行管理。



图 3-2　客户端管理程序



图 3-3 课程记录管理

学生管理

学号	姓名	密码
130119	蔡妙纯	1301190301
130119	陈家良	1301190302
130119	陈均杰	1301190303
130119	陈俊锐	1301190304

学号 姓名 密码

导入 增加 更新 删除

图 3-4 学生记录管理

成绩管理

课程 C#程序设计基础

学号	姓名	课程	成绩
130119	蔡妙纯	C#程序设计基础	90
130119	陈家良	C#程序设计基础	40
130119	陈均杰	C#程序设计基础	20

导入成绩 提交成绩

图 3-5 成绩记录管理

微信平台部分为课程成绩查询申请了一个微信公众号，学生用户关注公众号后凭自己的学号与查询密码查询成绩，命令格式是

学号，密码

输入学号与密码后发送给微信服务器，就能查询到自己的成绩，如果学号或者密码不正确，会收到一条错误信息。

学生也可以修改查询密码，命令格式是

学号，旧密码，新密码

输入学号、旧密码、新密码后发送给微信服务器，就能修改自己的密码，如图 3-6

所示。



图 3-6 微信成绩查询

3.1 课程记录管理

3.1.1 案例展示

设计一个 Web 网站，它包含一个名称为 MarkService.asmx 的 Web Service，这个服务管理一组课程，每门课程都有一个唯一的 ID 号和一个课程名称。同时设计一个 WPF 的客户端 MarkClient，它访问该服务，并能实现课程的增加、更新、删除等课程管理操作，如图 3-7 所示。

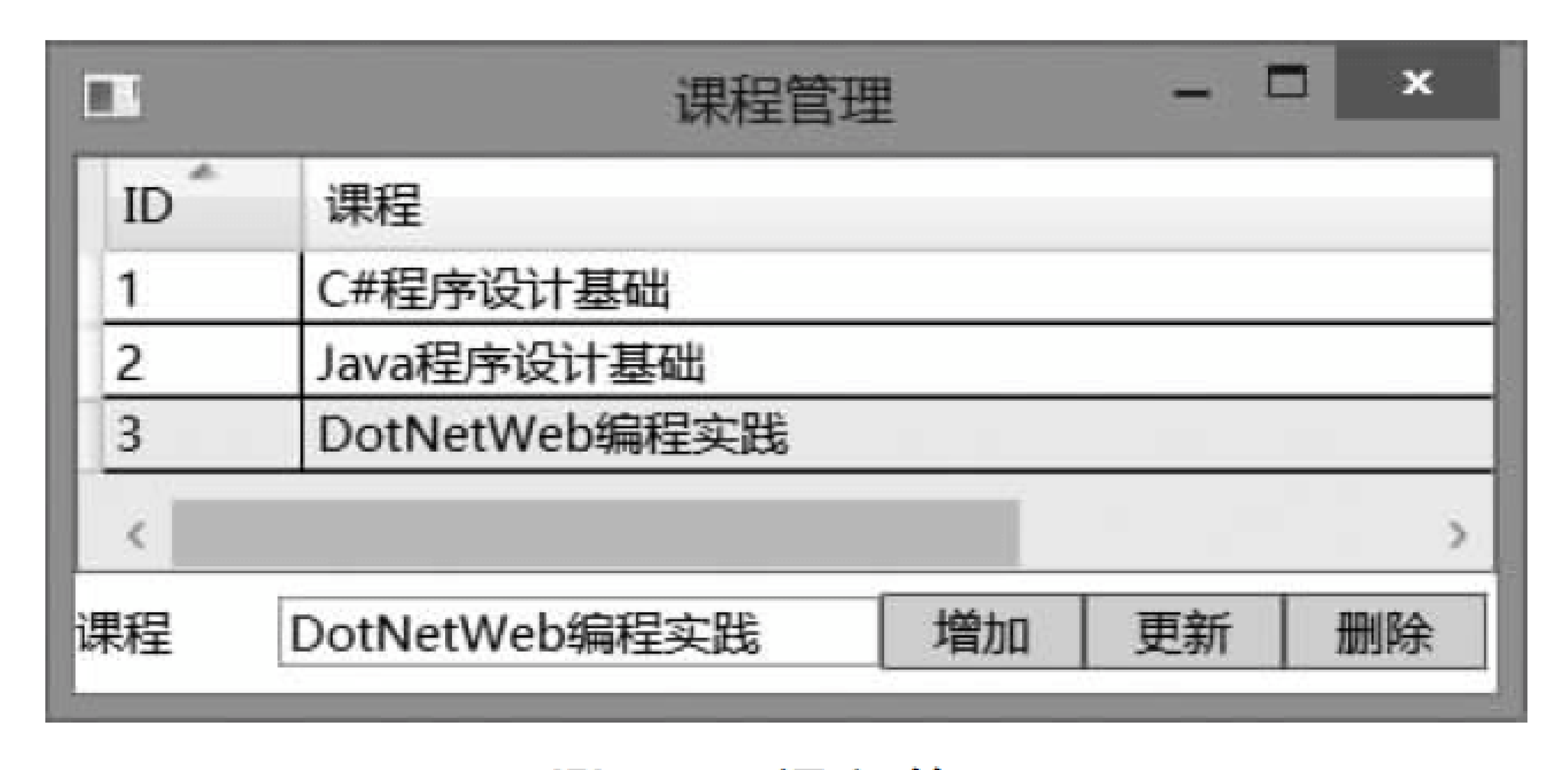


图 3-7 课程管理

3.1.2 技术要点

1. 数据库表

课程记录存储在数据库中，在 SQL Server 中建立数据库，然后在数据库中建立 subjects 课程表，这个表包含一个自动增长的 ID 号字段、课程名称字段 sSubject，SQL 命令如下：

```
create table subjects
(
    sID int identity(1,1) primary key,
    sSubject varchar(128)
)
```

这个 sID 自动成为每个课程的唯一编号及关键字，是确定一门课程的唯一标识。

2. Web Service 概述

Web Service(或称 Web 服务)是部署在 Web 网站上的一种特殊网页，这种网页不是给客户浏览的，而是给开发人员使用的。ASP. NET 的 Web Service 的文件扩展名称为“*.asmx”，它依赖 IIS 执行。

服务器的 Web Service 实际上是一组开放的接口函数，客户端可以调用这些函数。那么客户端怎样知道服务器的 Web Service 有什么接口函数呢？这需要定义一套协议来实现，XML(XSD)、SOAP 和 WSDL 就是构成 Web Service 平台的三大协议。

1) XML(XSD)

Web Service 采用 HTTP 协议传输数据，采用 XML 格式封装数据，即 XML 中说明调用远程服务对象的哪一个方法，传递的参数是什么以及服务对象的返回结果是什么。

XML 虽然解决了数据表示的问题，但它没有定义一套标准的数据类型。例如整形数到底代表什么，16 位还是 32 位，这些细节对实现互操作性很重要。因此还需要 XSD (XML Schema)来解决这个问题，XSD 定义了一套标准的数据类型，并给出了一种语言来扩展这套数据类型，Web Service 平台就是用 XSD 来作为其数据类型系统的。

2) SOAP

Web Service 通过 HTTP 协议发送请求和接收结果时，发送的请求内容和结果内容都采用 XML 格式封装，并增加了一些特定的 HTTP 消息头，这些特定的 HTTP 消息头和 XML 内容格式就是 SOAP 协议。可以理解为 SOAP 协议就是 HTTP 协议与 XML 数据的组合，SOAP 协议定义了 SOAP 消息的格式，它是基于 HTTP 协议的，也是基于 XML 和 XSD 的。简单来讲，SOAP 封装了服务器的接口函数的函数名称、函数参数与类型、返回结果类型等信息。

3) WSDL

Web Service 客户端要调用一个 Web Service 服务，首先要知道这个服务的地址在哪儿，并知道这个服务里有什么函数可以调用。所以 Web Service 服务器端首先要通过一

个 WSDL 文件来说明服务器里有哪些函数可以被外部调用，服务中有哪些方法，方法接受的参数是什么，返回值是什么，这些信息都必须告诉调用者。WSDL(Web Services Description Language)就是这样一个基于 XML 的语言，用于描述 Web Service 及其函数、参数和返回值，它是 Web Service 客户端和服务器端都能理解的标准格式。

微软公司的 Visual Studio 能够自动生成 WSDL 信息，同时在客户端能自动生成 Web Service 的代理类代码，这些功能使得开发 Web Service 的服务器与客户端都变得十分简单，图 3-8 为 Web Service 服务器与客户端程序的结构，它们之间通过 SOAP 协议在 Internet 网上传输数据与指令。

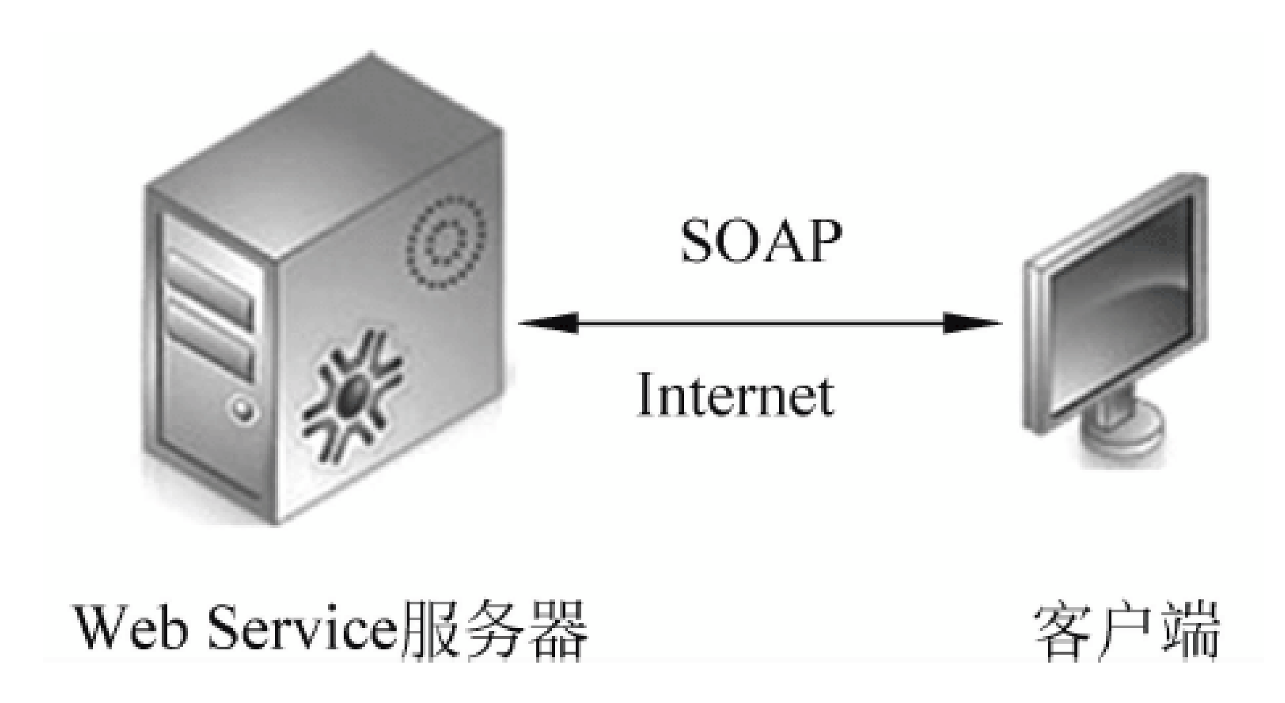


图 3-8 Web Service 程序结构

3. Web Service

在 ASP.NET 网站中添加 Web Service 很简单，选择网站后执行“添加新项”命令，弹出“添加新项”对话框后，选择“Web 服务(ASMX)”，输入服务的名称，例如输入 MarkService.asmx，然后单击“添加”按钮，如图 3-9 所示。



图 3-9 增加 Web Service

Visual Studio 会自动为这个服务 MarkService.asmx 生成如下的代码：

```
<%@Web Service Language="C# " Class="MarkService" %>
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
```

```
[Web Service(Namespace="http://tempuri.org/")]
[Web ServiceBinding(ConformsTo=WsiProfiles.BasicProfile1_1)]
public class MarkService: System.Web.Services.Web Service
{
    [WebMethod]
    public String HellowWorld()
    {
        return "Hellow World!";
    }
}
```

其中服务类 MarkService 从系统的 Web Service 类派生，类中有一个函数 HelloWorld，该函数返回一个字符串。HelloWorld 函数有两个重要的特征，一个是函数声明为 public，另一个是函数的前面有一个说明[WebMethod]，有这样两个特征的函数就是一个接口函数，是客户端能够识别与调用的函数。

在类 MarkService 的前面有一个特性说明：

```
[Web Service(Namespace="http://tempuri.org/")]
```

该特性说明服务的命名空间是 http://tempuri.org/，这个命名空间是可以更改的，一般建议设置为网站的地址，以保证其唯一性。

另外还有一个特性说明：

```
[Web ServiceBinding(ConformsTo=WsiProfiles.BasicProfile1_1)]
```

这个特性说明 Web Service 采用的绑定协议，一般为 BasicHttpBinding，这个特性不用修改。

4. 服务器 3 层结构

服务器程序还是采用 3 层结构，包括 3 个类项目 MarkModel、MarkDAL、MarkBLL 及一个 web 网站，如图 3-10 所示。

1）MarkModel

这是数据模型类库，在其中设计一个 SubjcetClass 类与数据库表 Subjects 对应，这个类有 sID、sSubject 属性。

```
public class SubjectClass
{
    public int sID { get; set; }
    public String sSubject { get; set; }
}
```

2）MarkDAL

这是数据访问层，其中包含 DBHelper 类来

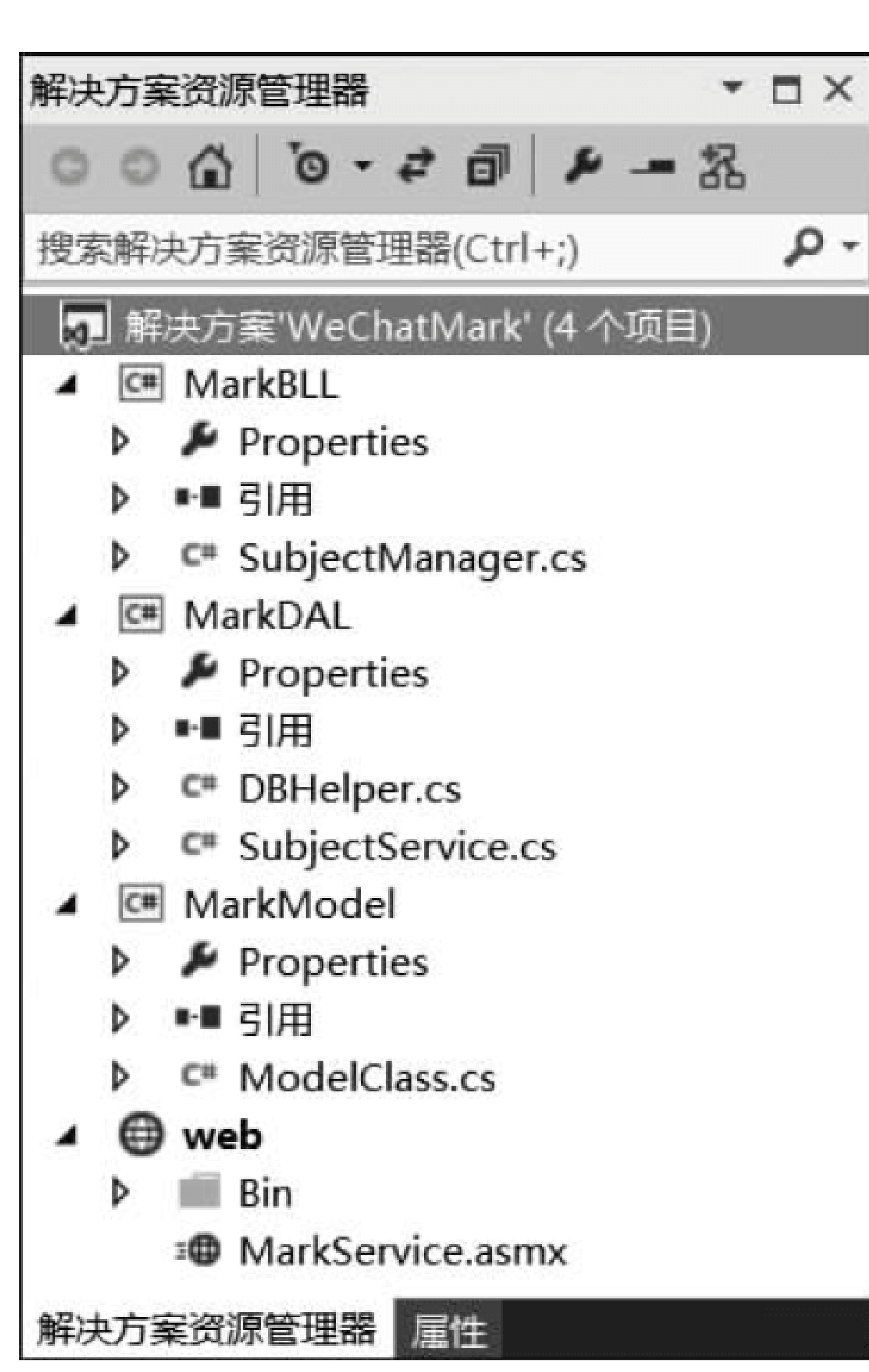


图 3-10 服务器结构

完成数据库的基本操作。设计 SubjectService 类实现课程的管理,其中 getSubjectDataTable 函数用来获取课程的数据表,另外的 addSubject、updateSubject、deleteSubject 函数分别实现课程的增加、更新、删除操作,每个函数的变量都采用 SubjectClass 类变量负责传递最新的课程数据。

```
namespace MarkDAL
{
    public class SubjectService
    {
        public int addSubject(ref SubjectClass s)
        {
            int ID=0;
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                SqlParameter pSubject=new SqlParameter { ParameterName=
                "@sSubject", SqlDbType=SqlDbType.Char, Value=s.sSubject };
                if(DB.executeCommand("insert into subjects (sSubject) values
                (@sSubject)", pSubject)>0)
                {
                    ID=DB.getScalar("select top 1 sID from subjects order by
                    sID desc");
                    s.sID=ID;
                }
                con.Close();
            }
            return ID;
        }
        public bool deleteSubject(SubjectClass s)
        {
            bool flag=false;
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                SqlParameter pID = new SqlParameter { ParameterName ="@sID",
                SqlDbType=SqlDbType.Int, Value=s.sID };
                if(DB.executeCommand("delete from subjects where sID=@sID",
                pID)>0) flag=true;
                con.Close();
            }
            return flag;
        }
        public bool updateSubject(SubjectClass s)
        {
            bool flag=false;
```

```
                using(SqlConnection con=new SqlConnection(DBHelper.conString))
                {
                    DBHelper DB=new DBHelper(con);
                    SqlParameter pID = new SqlParameter { ParameterName ="@sID",
                    SqlDbType=SqlDbType.Int, Value=s.sID };
                    SqlParameter pSubject=new SqlParameter { ParameterName=
                    "@sSubject", SqlDbType=SqlDbType.Char, Value=s.sSubject };
                    if(DB.executeCommand("update subjects set sSubject=@sSubject
                    where sID=@sID", pSubject,pID)>0) flag=true;
                    con.Close();
                }
                return flag;
            }
            public DataTable getSubjectDataTable()
            {
                DataTable dt=new DataTable("Subjects");
                using(SqlConnection con=new SqlConnection(DBHelper.conString))
                {
                    DBHelper DB=new DBHelper(con);
                    SqlDataAdapter adapter=DB.getAdapter("select * from subjects
                    order by sSubject");
                    adapter.Fill(dt);
                    con.Close();
                }
                return dt;
            }
        }
    }
```

3）MarkBLL

这是业务逻辑层,定义 SubjectManager 类进一步架起与 MarkDAL 层的桥梁,由于业务简单,所以这个层的函数基本上都是直接调用 MarkDAL 层的对应函数,这里不再列出。

3.1.3 服务器程序

在 MarkService. asmx 中增加 addSubject、updateSubbject、deleteSubject 函数实现课程的增加、更新、删除,同时 getSubjectDataTable 函数获取课程的数据集,程序如下:

```
<%@WebService Language="C# " Class="MarkService" %>
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Data;
using MarkModel;
```

```
using MarkBLL;
[WebService(Namespace="http://tempuri.org/")]
[WebServiceBinding(ConformsTo=WsiProfiles.BasicProfile1_1)]
public class MarkService  : System.Web.Services.WebService
{
    [WebMethod]
    public DataTable getSubjectDataTable()
    {
        SubjectManager manager=new SubjectManager();
        return manager.getSubjectDataTable();
    }
    [WebMethod]
    public bool updateSubject(SubjectClass s)
    {
        SubjectManager manager=new SubjectManager();
        return manager.updateSubject(s);
    }
    [WebMethod]
    public int addSubject(SubjectClass s)
    {
        SubjectManager manager=new SubjectManager();
        return manager.addSubject(ref s);
    }
    [WebMethod]
    public bool deleteSubject(SubjectClass s)
    {
        SubjectManager manager=new SubjectManager();
        return manager.deleteSubject(s);
    }
}
```

这个程序编写好后，在浏览器上调试执行，就可以看到图 3-11 所示的结果，这个页面上显示出这几个函数是可以被客户端识别的，即是公开的接口函数。

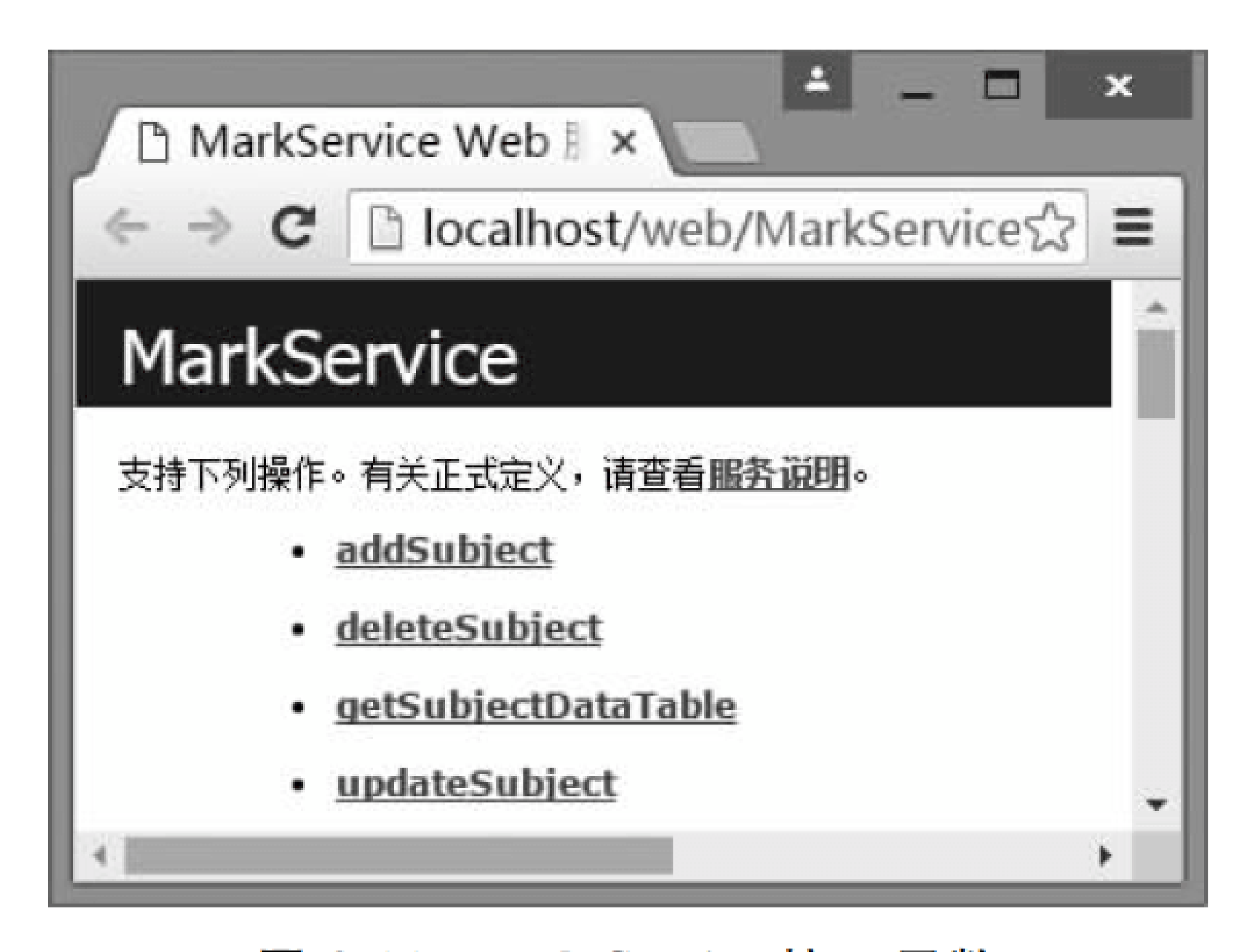


图 3-11 Web Service 接口函数

3.1.4 客户端程序

1. 界面设计

客户端采用 WPF 程序结构，界面部分主要包含一个 DataGrid 控件用来显示课程数据集，另外还放置了增加、删除、更新的按钮，XAML 代码如下：

```
<Window x:Class="MarkClient.MainWindow" x:Name="mainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="课程管理" Height="222.427" Width="367.66" Loaded="mainWindow_
        Loaded">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="30" />
        </Grid.RowDefinitions>
        <DataGrid x:Name="mGrid" Grid.Row="0" AutoGenerateColumns="False"
        SelectionMode="Single" IsReadOnly="True"  >
            <DataGrid.Columns>
                <DataGridTextColumn  Binding="{Binding sID}"  Header="ID"
                Width="50" />
                <DataGridTextColumn  Binding="{Binding sSubject}"  Header=
               "课程"  />
            </DataGrid.Columns>
        </DataGrid>
        <StackPanel Orientation="Horizontal" Grid.Row="1">
            <TextBlock Text="课程" Width="50" VerticalAlignment="Center" />
            <TextBox x:Name="txtSubject" Width="150" VerticalAlignment=
           "Center" />
            <Button x:Name="btAdd" Content="增加" Width="50"
            VerticalAlignment="Center" Click="btAdd_Click" />
            <Button x:Name="btUpdate" Content="更新"  Width="50"
            VerticalAlignment="Center" Click="btUpdate_Click"/>
            <Button x:Name="btDelete" Content="删除"  Width="50"
            VerticalAlignment="Center" Click="btDelete_Click"/>
        </StackPanel>
    </Grid>
</Window>
```

2. 代码设计

1）添加服务引用

客户端要调用服务器的 Web Service 接口函数，首先要找到这些函数的函数名称、函

数参数等信息。在 Visual Studio 中执行“项目”→“添加服务引用”命令，弹出“添加服务引用”对话框，在地址中输入服务器的地址，单击“转到”按钮客户端就可以发现服务器的 Web Service 服务，这个服务有 4 个接口函数。接下来在命名空间中输入一个名称，例如输入 WS 作为命名空间的名称，如图 3-12 所示。随后单击“确定”按钮关闭对话框。

图 3-12　添加服务引用

添加服务引用的过程实际上就是客户端为服务器函数生成本地的代理函数的过程，客户端调用服务器函数时只要调用本地的代理函数就可以了，代理函数负责与服务器交互，这样开发客户端程序就变得十分简单。

2）调用服务器函数

在客户端首先要建立一个访问服务器的对象 client，这个对象定义如下：

```
WS.MarkServiceSoapClient client;
```

其中 WS 是在前面定义的命名空间的名称，这个空间中有一个名为 MarkServiceSoapClient 的类，这个类是客户端在添加服务引用时自动生成的代理类，这个类中包含了服务器的 4 个接口函数的代理函数，而且名称与服务器的函数一样，只要调用这个类的对应函数，就等同于调用了服务器的函数。为客户端编写程序代码如下：

```
public partial class MainWindow : Window
{
    WS.MarkServiceSoapClient client;
    DataTable dt;
    DataView dv;
    public MainWindow()
    {
```

```
        InitializeComponent();
    }
    void showMsg(String s)
    {
        MessageBox.Show(s, "Information", MessageBoxButton.OK);
    }
    bool confirm(String s)
    {
        return(MessageBox.Show(s, "Confirmation", MessageBoxButton.YesNo)==
        MessageBoxResult.Yes);
    }
    private void btAdd_Click(object sender, RoutedEventArgs e)
    {
        String s=txtSubject.Text.Trim();
        if(s!="")
        {
            try
            {
                int sID = client. addSubject (new WS. SubjectClass { sID = 0,
                sSubject=s });
                if(sID>0)
                {
                    DataRow row=dt.NewRow();
                    row["sID"]=sID;
                    row["sSUbject"]=s;
                    dt.Rows.Add(row);
                    dt.AcceptChanges();
                }
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
    private void btUpdate_Click(object sender, RoutedEventArgs e)
    {
        String s=txtSubject.Text.Trim();
        int index=mGrid.SelectedIndex;
        if(s !="" && index>=0)
        {
            if(confirm("确实要更新?"))
            {
                try
                {
                    if(client.updateSubject(new WS.SubjectClass { sID=(int)dv
                    [index]["sID"], sSubject=s }))
```

```
                    {
                        dv[index]["sSubject"]=s;
                        dt.AcceptChanges();
                    }
                }
                catch(Exception exp) { showMsg(exp.Message); }
            }
        }
    }
    private void btDelete_Click(object sender, RoutedEventArgs e)
    {
        int index=mGrid.SelectedIndex;
        if(index>=0)
        {
            if(confirm("删除该课程?"))
            {
                try
                {
                    if(client.deleteSubject(new WS.SubjectClass { sID=(int)dv
                    [index]["sID"], sSubject=dv[index]["sSubject"].ToString
                    () }))
                    {
                        dv[index].Delete();
                        dt.AcceptChanges();
                    }
                }
                catch(Exception exp) { showMsg(exp.Message); }
            }
        }
    }
    private void mainWindow_Loaded(object sender, RoutedEventArgs e)
    {
        try
        {
            client=new WS.MarkServiceSoapClient();
            dt=client.getSubjectDataTable();
            dv=dt.DefaultView;
            mGrid.ItemsSource=dv;
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
}
```

程序中使用一个 DataTable 对象 dt 来存储从服务器获取的课程数据集，这个数据集

在程序窗体装载时就获取，然后通过它的 DataView 对象绑定到 DataGrid 上，显示出课程信息。

3.1.5 拓展训练

如果服务器端有错误出现，此时服务器会抛出一个错误，而且客户端能捕捉到这个错误。读者可以尝试把数据库的 subjects 表改成另外一个名字，此时服务器会抛出找不到 subjects 表的错误，客户端运行时可以看到这个错误的信息，如图 3-13 所示。



图 3-13 错误信息

由此可见，客户端不但可以调用 Web Service 服务器的函数，还能捕捉到服务器的异常，这些都是本地代理的功劳，本地代理极大地简化了客户端的程序。有兴趣的读者可以在解决方案资源管理器中单击 WS 打开对象管理器，查看 WS 空间的各个类的定义，它们就是本地的代理。

3.2 学生记录管理

3.2.1 案例展示

下面扩展服务器端的功能，使得它能够管理学生记录，学生信息包括学号、姓名、密码。再设计客户端程序管理学生的记录，输入学生的学号、姓名、密码信息后单击“增加”按钮就增加一个学生，单击“更新”就更新该学生的信息，选择一条学生记录后单击“删除”按钮就删除该学生，如图 3-14 所示。



图 3-14 学生管理

3.2.2 技术要点

1. 数据库表

在数据库中建立学生信息表 students，包括学号 sNo、姓名 sName 以及学生密码 sPass 信息，SQL 命令如下：

```
create table students
(
    sNo varchar(32) primary key,
    sName varchar(32),
    sPass varchar(256)
)
```

在 MarkModel 中设计 StudentClass 学生类与这个表对应：

```
public class StudentClass
{
    public String sNo { get; set; }
    public String sName { get; set; }
    public String sPass { get; set; }
}
```

2. 学生管理

在 MarkDAL 中增加 StudentService 的类文件，设计增加学生函数 addStudent、更新学生函数 updateStudent、删除学生函数 deleteStudent，这些函数都以 StudentClass 对象作为函数参数。在管理学生记录时以 sNo 为关键字查找学生记录。为了在客户端能获取学生记录，另外设计获取学生数据集函数 getStudentdataTable，这个函数没有参数。

```
namespace MarkDAL
{
    public class StudentService
```

```
{
    public bool addStudent(StudentClass s)
    {
        bool flag=false;
        using(SqlConnection con=new SqlConnection(DBHelper.conString))
        {
            DBHelper DB=new DBHelper(con);
            SqlParameter pNo = new SqlParameter { ParameterName =" @ sNo",
            SqlDbType=SqlDbType.Char, Value=s.sNo };
            SqlParameter pName=new SqlParameter { ParameterName="@sName",
            SqlDbType=SqlDbType.Char, Value=s.sName };
            SqlParameter pPass=new SqlParameter { ParameterName="@sPass",
            SqlDbType=SqlDbType.Char, Value=s.sPass };
            try
            {
                if(DB.executeCommand("insert into students (sNo, sName,
                sPass) values (@sNo,@sName,@sPass)", pNo, pName, pPass)>
                0) flag=true;
            }
            catch { }
            con.Close();
        }
        return flag;
    }
    public bool deleteStudent(StudentClass student)
    {
        bool flag=false;
        using(SqlConnection con=new SqlConnection(DBHelper.conString))
        {
            DBHelper DB=new DBHelper(con);
            SqlParameter pNo = new SqlParameter { ParameterName =" @ sNo",
            SqlDbType=SqlDbType.Char, Value=student.sNo };
            if(DB.executeCommand("delete from students where sNo = @ sNo",
            pNo)>0)  flag=true;
            con.Close();
        }
        return flag;
    }
    public bool updateStudent(StudentClass student)
    {
        bool flag=false;
        using(SqlConnection con=new SqlConnection(DBHelper.conString))
        {
            DBHelper DB=new DBHelper(con);
```

```
                SqlParameter pNo=new SqlParameter { ParameterName="@sNo",
                SqlDbType=SqlDbType.Char, Value=student.sNo };
                SqlParameter pName=new SqlParameter { ParameterName=
                "@sName", SqlDbType=SqlDbType.Char, Value=student.sName };
                SqlParameter pPass=new SqlParameter { ParameterName=
                "@sPass", SqlDbType=SqlDbType.Char, Value=student.sPass };
                if(DB.executeCommand("update students set sName=@sName,sPass
                =@sPass where sNo=@sNo", pNo, pName,pPass)>0) flag=true;
                con.Close();
            }
            return flag;
        }
        public DataTable getStudentDataTable()
        {
            DataTable dt=new DataTable("students");
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                SqlDataAdapter adapter=DB.getAdapter("select * from students
                order by sNo");
                adapter.Fill(dt);
                con.Close();
            }
            return dt;
        }
    }
```

在 MarkBLL 中定义 StudentManager 类，进一步架起与 MarkDAL 层的桥梁，由于业务简单，所以这个层的函数基本上都是直接调用 MarkDAL 层的对应函数，这里不再列出。

3.2.3 服务器端程序

在 MarkService.asmx 中增加 addStudent、updateStudent、deleteStudent 函数实现学生记录的增加、更新、删除，同时 getStudentDataTable 函数获取课程的数据集，它们都是直接调用 MarkBLL 中 StudentManager 类的对应函数，这些函数又调用 MarkDAL 中 StudentService 的对应函数，程序如下：

```
<%@WebService Language="C# " Class="MarkService" %>
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Data;
```

```
using MarkModel;
using MarkBLL;
[WebService(Namespace="http://tempuri.org/")]
[WebServiceBinding(ConformsTo=WsiProfiles.BasicProfile1_1)]
public class MarkService  : System.Web.Services.WebService
{
    [WebMethod]
    public DataTable getStudentDataTable()
    {
        StudentManager manager=new StudentManager();
        return manager.getStudentDataTable();
    }
    [WebMethod]
    public bool updateStudent(StudentClass s)
    {
        StudentManager manager=new StudentManager();
        return manager.updateStudent(s);
    }
    [WebMethod]
    public bool addStudent(StudentClass s)
    {
        StudentManager manager=new StudentManager();
        return manager.addStudent(s);
    }
    [WebMethod]
    public bool deleteStudent(StudentClass s)
    {
        StudentManager manager=new StudentManager();
        return manager.deleteStudent(s);
    }
}
```

3.2.4 客户端程序

1. 界面设计

客户端界面主要包含一个 DataGrid 显示学生数据集，另外有 3 个 TextBox 用于输入学号、姓名、密码，3 个 Button 用于实现增加、删除、更新的按钮操作。

```
<Window x:Class="MarkClient.MainWindow" x:Name="mainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="学生管理" Height="222.427" Width="331.94" Loaded="mainWindow_
        Loaded">
    <Grid>
```

```
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="30" />
            <RowDefinition Height="30" />
        </Grid.RowDefinitions>
        <DataGrid x:Name="mGrid" Grid.Row="0" AutoGenerateColumns="False"
        SelectionMode="Single" IsReadOnly="True">
            <DataGrid.Columns>
                <DataGridTextColumn  Binding="{Binding sNo}"  Header="学号"
                Width="50"  />
                <DataGridTextColumn  Binding="{Binding sName}"  Header="姓
                名"  Width="100" />
                <DataGridTextColumn  Binding="{Binding sPass}"  Header="密
                码"  Width="100"  />
            </DataGrid.Columns>
        </DataGrid>
        <StackPanel Orientation="Horizontal" Grid.Row="1">
            <TextBlock Text="学号" VerticalAlignment="Center" />
            <TextBox x:Name="txtNo" Text="" Width="80" VerticalAlignment=
           "Center" />
            <TextBlock Text="姓名" VerticalAlignment="Center" />
            <TextBox x:Name="txtName" Text="" Width="80" VerticalAlignment=
           "Center" />
            <TextBlock Text="密码" VerticalAlignment="Center" />
            <TextBox x:Name="txtPass" Text="" Width="80" VerticalAlignment=
           "Center" />
        </StackPanel>
        <StackPanel Orientation="Horizontal" Grid.Row="2">
            <Button x:Name="btAdd" Content="增加" Width="50"
            VerticalAlignment="Center" Click="btAdd_Click" />
            <Button x:Name="btUpdate" Content="更新"  Width="50"
            VerticalAlignment="Center" Click="btUpdate_Click"/>
            <Button x:Name="btDelete" Content="删除" Width="50"
            VerticalAlignment="Center"  Click="btDelete_Click"/>
        </StackPanel>
    </Grid>
</Window>
```

2. 代码设计

在程序启动时就获取学生记录数据集，并显示在 DataGrid 控件上。学号 sNo 是学生记录的关键字，因此在增加学生时先在客户端对学号进行检测，如果已经有相同学号的学生记录，就不再增加该学生。为了简单起见，程序采用同步操作的函数与服务器进行通信。

```
public partial class MainWindow : Window
{
    WS.MarkServiceSoapClient client;
    DataTable dt;
    DataView dv;
    public MainWindow()
    {
        InitializeComponent();
    }
    void showMsg(String s)
    {
        MessageBox.Show(s, "Information", MessageBoxButton.OK);
    }
    bool confirm(String s)
    {
        return(MessageBox.Show(s, "Confirmation", MessageBoxButton.YesNo)==
        MessageBoxResult.Yes);
    }
    private void btAdd_Click(object sender, RoutedEventArgs e)
    {
         String sNo=txtNo.Text.Trim(),sName=txtName.Text.Trim(),sPass=
        txtPass.Text.Trim();
        if(sNo!="" &&sName!="")
        {
            DataRow row=dt.AsEnumerable().FirstOrDefault(x=>x["sNo"].
            ToString()==sNo);
            if(row!=null)
            {
                showMsg(sNo+"已经存在!"); return;
            }
            if(sPass=="") sPass=sNo;
            try
            {
                if(client.addStudent(new WS.StudentClass { sNo=sNo,sName=
               sName,sPass=sPass}))
                {
                    row=dt.NewRow();
                    row["sNo"]=sNo;
                    row["sName"]=sName;
                    row["sPass"]=sPass;
                    dt.Rows.Add(row);
                    dt.AcceptChanges();
                }
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
```

```
}
private void btUpdate_Click(object sender, RoutedEventArgs e)
{
    String  sName=txtName.Text.Trim(), sPass=txtPass.Text.Trim();
    int index=mGrid.SelectedIndex;
    if(sName !="" && index>=0)
    {
        String sNo=dv[index]["sNo"].ToString();
        if(sPass=="") sPass=sNo;
        try
        {
            if(client.updateStudent(new WS.StudentClass { sNo= sNo, sName
            =sName, sPass=sPass }))
            {
                dv[index]["sName"]=sName;
                dv[index]["sPass"]=sPass;
                dt.AcceptChanges();
            }
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
}
private void btDelete_Click(object sender, RoutedEventArgs e)
{
    int index=mGrid.SelectedIndex;
    if(index>=0)
    {
        if(confirm("删除该学生?"))
        {
            try
            {
                if(client.deleteStudent(new WS.StudentClass { sNo=dv
                [index]["sNo"].ToString(),sName="",sPass="" }))
                {
                    dv[index].Delete();
                    dt.AcceptChanges();
                }
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
}
private void mainWindow_Loaded(object sender, RoutedEventArgs e)
{
    try
    {
```

```
                client=new WS.MarkServiceSoapClient();
                dt=client.getStudentDataTable();
                dv=dt.DefaultView;
                mGrid.ItemsSource=dv;
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
```

3.2.5 拓展训练

在实际应用中一般需要把一批学生记录导入到服务器,而不是像目前这个程序那样一个一个地将学生增加到服务器。一般来说学生名单常常是一个 Excel 文件,第一列是学号,第二列是姓名。这个 Excel 文件可以另存为 CSV 文件。CSV 文件是一个文本文件,文件中同一行的不同数据用逗号隔开,这个 CSV 文件的格式如下:

```
学号1,姓名1
学号2,姓名2
⋮
```

这样的文件数据是很容易导入到系统中的。

1. 服务器设计

在服务器的 MarkDAL 的 StudentService 中增加一个 appendStudentList 的函数,这个函数负责批量增加学生记录:

```
public int appendStudentList(List<StudentClass>students)
{
    int count=0;
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        foreach (StudentClass s in students)
        {
            SqlParameter pNo = new SqlParameter { ParameterName =" @ sNo",
            SqlDbType=SqlDbType.Char,Value=s.sNo};
            SqlParameter pName = new SqlParameter { ParameterName =" @ sName",
            SqlDbType=SqlDbType.Char,Value=s.sName};
            SqlParameter pPass = new SqlParameter { ParameterName =" @ sPass",
            SqlDbType=SqlDbType.Char, Value=s.sPass };
            try
            {
                //有些记录已经在数据库表中,插入该记录会失败
                if(DB.executeCommand("insert into students (sNo,sName,sPass)
```

```
                values (@sNo,@sName,@sPass)", pNo, pName,pPass)>0)++count;
            }
            catch { }
        }
        con.Close();
    }
    return count;
}
```

函数接受一个 List<StudentClass>的列表参数，它是要增加的一批学生的对象列表。在增加的时候，如果一个学生已经存在，就不再增加；如果不存在，就增加到 students 表中。

同时在 MarkBLL 的 StudentManager 中也增加一个类似的 appendStudentList 函数，它调用 StudentService 中对应的 appendStudentList 函数，在 MarService.asmx 中增加 appendStudentList 的接口函数：

```
[WebMethod]
public int appendStudentList(List<StudentClass>students)
{
    StudentManager manager=new StudentManager();
    return manager.appendStudentList(students);
}
```

2. 客户端设计

在客户端再增加一个导入按钮：

```
<Button x:Name="btLoad" Content="导入" Width="50" VerticalAlignment=
"Center" Click="btLoad_Click" />
```

编写这个按钮的事件函数，执行时选择磁盘中一个 CSV 文件，这个 CSV 文件的数据格式是一个学生信息占一行，读出一行，用逗号分开两个数据，前面一个为学号，后面一个为姓名，学生的密码初始值设置为学号，就可以生成 List<WS.StudentClass>列表 students，调用服务器的 appendStudentList 函数就完成批量导入学生名单到服务器的工作。学生名单批量导入后再次刷新学生数据集，重新绑定到 DataGrid 中显示。

```
private void btLoad_Click(object sender, RoutedEventArgs e)
{
    OpenFileDialog dlg=new OpenFileDialog();
    dlg.Filter="csv|*.csv";
    if((bool)dlg.ShowDialog())
    {
        List<WS.StudentClass>students=new List<WS.StudentClass>();
        StreamReader sr=new StreamReader(dlg.FileName, Encoding.Default);
        String s="";
```

```
            while((s=sr.ReadLine())!=null)
            {
                String[] st=s.Split(new char[] { ',' }, StringSplitOptions.
                RemoveEmptyEntries);
                if(st.Length==2) students.Add(new WS.StudentClass { sNo=st[0],
                sName=st[1], sPass=st[0] });
            }
            sr.Close();
            try
            {
                if(client.appendStudentList(students.ToArray())>0)
                {
                    dt=client.getStudentDataTable();
                    dv=dt.DefaultView;
                    mGrid.ItemsSource=dv;
                }
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
```

3.3 成绩记录管理

3.3.1 案例展示

每个学生都有各门课程的成绩，在服务器端设计管理成绩的功能，并设计 Web Service 接口函数，在客户端调用这些接口函数就可以实现对成绩的管理，如图 3-15 所示。在客户端启动后得到所有的课程列表，放在一个下拉列表框中，当选择一门课程后就显示出每个学生该门课程的成绩。客户端在成绩一列可以输入每个学生的成绩，输入完成后单击“提交成绩”按钮就把成绩提交到服务器保存。



图 3-15 成绩管理

3.3.2 技术要点

1. 数据库表

在数据库中建立学生成绩表 marks,SQL 命令如下:

```
create table marks
(
  sNo varchar(32) not null foreign key references students (sNo) on delete
  cascade,
  sID int not null foreign key references subjects (sID) on delete cascade,
  mMark int,
  constraint mark_pk primary key (sNo,sID)
)
```

这个表包含学号 sNo、课程编号 sID、成绩 mMark,其中 sNo 外键参照 students 的 sNo 字段,sID 外键参照 subjets 表的 sID 字段。组合(sNo,sID)成为关键字,因为一个学生可以有很多门课程,每门课程只有一个成绩。

在 MarkModel 中设计 MarkClass 类与这个表的各个字段对应:

```
public class MarkClass
{
    public String sNo { get; set; }
    public int sID { get; set; }
    public int mMark { get; set; }
}
```

2. 成绩管理

在 MarkDAL 中设计 MarkService 类文件,再设计 getMarkDataTable 函数获取 sID 课程编号的学生成绩数据集。在客户端选择一门课程时,必须得到这门课程的学生成绩数据集,而这个数据集是从 marks 表中得到的,因此要设计 insertStudentList 函数,它是一个内部调用的函数,目的是在选择一门课程后,在 marks 表中生成这门课程的所有学生记录。updateMarkList 函数是更新学生成绩的函数,其中的参数是一个成绩列表 List<MarkClass>类型,它包含了所有学生的该门课程的成绩。

```
namespace MarkDAL
{
    public class MarkService
    {
        public int updateMarkList(List<MarkClass>marks)
        {
            int count=0;
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
```

```
        {
            DBHelper DB=new DBHelper(con);
            foreach (MarkClass m in marks)
            {
                SqlParameter pNo=new SqlParameter { ParameterName="@sNo",
                SqlDbType=SqlDbType.Char, Value=m.sNo };
                SqlParameter pID=new SqlParameter { ParameterName="@sID",
                SqlDbType=SqlDbType.Int, Value=m.sID };
                SqlParameter pMark=new SqlParameter { ParameterName=
                "@mMark", SqlDbType=SqlDbType.Int, Value=m.mMark };
                if(DB.executeCommand("update marks set mMark=@mMark where
                sNo=@sNo and sID=@sID", pMark, pNo, pID)>0)++count;
            }
            con.Close();
        }
        return count;
    }
    void insertStudentList(DBHelper DB,SqlParameter pID)
    {
        SqlDataAdapter adapter= DB.getAdapter("select sNo from students
        where sNo not in(select sNo from marks where sID=@sID)", pID);
        DataTable dt=new DataTable();
        adapter.Fill(dt);
        foreach(DataRow row in dt.Rows)
        {
            SqlParameter pNo= new SqlParameter { ParameterName="@sNo",
            SqlDbType=SqlDbType.Char, Value=row["sNo"] };
            DB.executeCommand("insert into marks (sNo,sID,mMark) values
            (@sNo,@sID,0)", pNo,pID);
        }
    }
    public DataTable getMarkDataTable(int sID)
    {
        DataTable dt=new DataTable("marks");
        using(SqlConnection con=new SqlConnection(DBHelper.conString))
        {
            DBHelper DB=new DBHelper(con);
            SqlParameter pID= new SqlParameter { ParameterName="@sID",
            SqlDbType=SqlDbType.Int, Value=sID };
            insertStudentList(DB, pID);
            String sql="select st.sNo as sNo,sName,sSubject,mMark from
            students st join marks m on st.sNo=m.sNo join subjects su on
            m.sID=su.sID and su.sID=@sID order by st.sNo,sSubject";
            SqlDataAdapter adapter=DB.getAdapter(sql, pID);
```

```
                adapter.Fill(dt);
                con.Close();
            }
            return dt;
        }
}
```

3. 成绩输入框

在客户端的成绩列是一个可编辑的列，因此采用<DataGridTemplateColumn>模板来设置该列。该列的单元数据处于显示状态时使用 TextBlock 控件直接显示，如果处于编辑状态，就变成一个名称为 txtMark 的可编辑的 TextBox 控件。这个列的结构如下：

```
<DataGridTemplateColumn x:Name="markColumn"  Header="成绩"  Width="100"
IsReadOnly="False">
    <DataGridTemplateColumn.CellTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding mMark}" />
        </DataTemplate>
    </DataGridTemplateColumn.CellTemplate>
    <DataGridTemplateColumn.CellEditingTemplate>
        <DataTemplate>
            <TextBox x:Name="txtMark" Text="{Binding mMark,Mode=TwoWay}"
                     PreviewKeyDown="txtMark_PreviewKeyDown" MaxLength="3"
                     TextChanged="txtMark_TextChanged" >
            </TextBox>
        </DataTemplate>
    </DataGridTemplateColumn.CellEditingTemplate>
</DataGridTemplateColumn>
```

在成绩输入时，为了让输入的成绩控制在 0～100 的范围内，设置 txtMark 的 MaxLength="3"控制最多可以输入 3 个字符，同时设置其 PreviewKeyDown 事件函数，这个函数控制 txtMark 中只能输入数字及退格键，输入其他键无效。

```
<private void txtMark_PreviewKeyDown(object sender, KeyEventArgs e)
{
    //成绩输入框值允许输入 0~9 的数字与退格
    if(e.Key>=Key.D0 && e.Key<=Key.D9||e.Key==Key.Back) e.Handled=false;
    else e.Handled=true;
}
```

另外还要设置 TextChanged 事件函数，这个函数随时监测输入的值，一旦成绩无效就给出警告，要求重新输入。

3.3.3 服务器程序

服务器程序 MarkService. asmx 中定义了获取课程数据集函数 getSubjectDataTable、获取指定课程的学生成绩表数据集函数 getSubjectDataTable、更新学生成绩的函数 updateMarkList。

```
public class MarkService  : System.Web.Services.WebService
{
    [WebMethod]
    public DataTable getSubjectDataTable()
    {
        SubjectManager manager=new SubjectManager();
        return manager.getSubjectDataTable();
    }
    [WebMethod]
    public DataTable getMarkDataTable(int sID)
    {
        MarkManager manager=new MarkManager();
        return manager.getMarkDataTable(sID);
    }
    [WebMethod]
    public int updateMarkList(List<MarkClass>marks)
    {
        MarkManager manager=new MarkManager();
        return manager.updateMarkList(marks);
    }
}
```

3.3.4 客户端程序

1. 界面设计

客户端程序包含一个 ComboBox 下拉列表，用来选择课程，这个列表变化时触发其事件 SelectionChanged，在对应函数中获取该课程的学生成绩数据集显示在 DataGrid 上。成绩录入框 txtMark 的 Binding 的 Mode 设置为 TwoWay，以便在文本框中输入的成绩能立即更新到它的数据集中。

```
<Window x:Class="MarkClient.MainWindow" x:Name="mainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="成绩管理" Height="222.427" Width="331.94" Loaded="mainWindow_
        Loaded">
    <Grid>
```

```
<Grid.RowDefinitions>
    <RowDefinition Height="30" />
    <RowDefinition Height="*" />
    <RowDefinition Height="30" />
</Grid.RowDefinitions>
<StackPanel Orientation="Horizontal" Grid.Row="0"
HorizontalAlignment="Center">
    <TextBlock Text="课程" VerticalAlignment="Center" />
    <ComboBox x:Name="cbSubject" Width="150" VerticalAlignment=
    "Center" SelectionChanged="cbSubject_SelectionChanged" />
</StackPanel>
<DataGrid x:Name="mGrid" Grid.Row="1" AutoGenerateColumns="False"
SelectionMode="Single" CanUserDeleteRows="False" CanUserAddRows=
"False" SelectionUnit="Cell" SelectedCellsChanged="mGrid_Selected-
CellsChanged">
    <DataGrid.Columns>
        <DataGridTextColumn  Binding="{Binding sNo}"  Header="学号"
        Width="50" IsReadOnly="True" />
        <DataGridTextColumn  Binding="{Binding sName}"  Header="姓
        名"  Width="100" IsReadOnly="True" />
        <DataGridTextColumn  Binding="{Binding sSubject}"
        Header="课程"  Width="100" IsReadOnly="True" />
        <DataGridTemplateColumn x:Name="markColumn"  Header="成绩"
        Width="100" IsReadOnly="False">
            <DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding mMark}" />
                </DataTemplate>
            </DataGridTemplateColumn.CellTemplate>
            <DataGridTemplateColumn.CellEditingTemplate>
                <DataTemplate>
                    <TextBox x:Name="txtMark" Text="{Binding mMark,
                    Mode=TwoWay}" PreviewKeyDown="txtMark_PreviewKey-
                    Down" MaxLength="3" TextChanged="txtMark_Text-
                    Changed" >
                    </TextBox>
                </DataTemplate>
            </DataGridTemplateColumn.CellEditingTemplate>
        </DataGridTemplateColumn>
    </DataGrid.Columns>
</DataGrid>
<StackPanel Orientation="Horizontal" Grid.Row="2"
HorizontalAlignment="Center">
    <Button x:Name="btUpdate" Content="提交成绩"  Width="50"
```

```
            Click="btUpdate_Click"/>
        </StackPanel>
    </Grid>
</Window>
```

2. 代码设计

程序中的 sdv 是课程数据集的视图对象，课程数据集包含课程的 sID 与 sSubject，下拉列表框负责显示课程名称，当选择一个课程后就确定出该课程的 sID，然后获取该课程的学生数据集 dt，把这个数据集显示在 DataGrid 上。

```
public partial class MainWindow : Window
{
    WS.MarkServiceSoapClient client;
    DataTable dt;
    DataView sdv, dv;
    public MainWindow()
    {
        InitializeComponent();
    }
    void showMsg(String s)
    {
        MessageBox.Show(s, "Information", MessageBoxButton.OK);
    }
    bool confirm(String s)
    {
        return(MessageBox.Show(s, "Confirmation", MessageBoxButton.YesNo)==
        MessageBoxResult.Yes);
    }
    private void btUpdate_Click(object sender, RoutedEventArgs e)
    {
        if(cbSubject.SelectedIndex>=0)
        {
            try
            {
                int sID=(int)sdv[cbSubject.SelectedIndex]["sID"];
                List<WS.MarkClass>marks=new List<WS.MarkClass>();
                foreach (DataRowView row in dv)
                {
                    WS.MarkClass m=new WS.MarkClass { sNo=row["sNo"].ToString(),
                    sID=sID, mMark=(int)row["mMark"] };
                    marks.Add(m);
                }
                int count=client.updateMarkList(marks.ToArray());
```

```
            showMsg("更新"+count.ToString()+"条记录!");
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
}
private void mainWindow_Loaded(object sender, RoutedEventArgs e)
{
    try
    {
        client=new WS.MarkServiceSoapClient();
        sdv=client.getSubjectDataTable().DefaultView;
        foreach (DataRowView row in sdv)
        {
            cbSubject.Items.Add(row["sSubject"].ToString());
        }
    }
    catch(Exception exp) { showMsg(exp.Message); }
}
private void cbSubject_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if(cbSubject.SelectedIndex>=0)
    {
        int sID=(int)sdv[cbSubject.SelectedIndex]["sID"];
        try
        {
            dt=client.getMarkDataTable(sID);
            dv=dt.DefaultView;
            mGrid.ItemsSource=dv;
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
    else mGrid.ItemsSource=null;
}
private void txtMark_PreviewKeyDown(object sender, KeyEventArgs e)
{
    //成绩输入框值允许输入 0~9 的数字与退格
    if(e.Key>=Key.D0 && e.Key<=Key.D9 || e.Key==Key.Back) e.Handled=
   false;
    else e.Handled=true;
}
private void txtMark_TextChanged(object sender, TextChangedEventArgs e)
{
    //输入值变化后,判断成绩是否有效,如果无效则会要求重新输入
```

```
            TextBox txt=(TextBox)sender;
            String s=txt.Text;
            int m=0;
            int.TryParse(s, out m);
            if(m<0 || m>100)
            {
                showMsg("成绩值无效!");
                if(s.Length==3) txt.Text=s.Substring(0, 2);
                txt.Focus();
            }
        }
        private void mGrid_SelectedCellsChanged(object sender,
        SelectedCellsChangedEventArgs e)
        {
            mGrid.BeginEdit();
        }
    }
```

程序中使用了 DataGrid 的 SelectedCellsChanged 事件函数，它是在选择的单元格发生变化时触发的事件，当鼠标选择了成绩单元时立即执行 mGrid.BeginEdit()命令，使得该单元立即进入编辑状态，这极大地方便了成绩输入。

3.3.5 拓展训练

类似批量导入学生名单一样，成绩也可以批量导入。如果成绩的 CSV 文件格式如下：

```
学号 1,姓名 1,成绩 1
学号 2,姓名 2,成绩 2
⋮
```

那么可以在客户端增加一个导入按钮，单击按钮时读取 CSV 文件的每一行，各部分按逗号分开，第一部分为学号，第二部分为姓名，第三部分为成绩，生成 List<WS.MarkClass> 的成绩列表 marks，调用 updateMarkList 函数就可以完成成绩更新，实现成绩批量上传到服务器，批量上传的函数如下：

```
    private void btLoad_Click(object sender, RoutedEventArgs e)
    {
        if(cbSubject.SelectedIndex>=0)
        {
            OpenFileDialog dlg=new OpenFileDialog();
            dlg.Filter="csv|*.csv";
            if((bool)dlg.ShowDialog())
            {
                int sID=(int)sdv[cbSubject.SelectedIndex]["sID"];
```

```
                List<WS.MarkClass>marks=new List<WS.MarkClass>();
                StreamReader sr=new StreamReader(dlg.FileName, Encoding.Default);
                String s="";
                while ((s=sr.ReadLine()) !=null)
                {
                    String[] st=s.Split(new char[] { ',' }, StringSplitOptions.
                    RemoveEmptyEntries);
                    if(st.Length==3)
                    {
                        int m=0;
                        int.TryParse(st[2], out m);
                        if(m<0 || m>100) m=0;
                        marks.Add(new WS.MarkClass { sNo=st[0], sID=sID, mMark=m });
                    }
                }
                sr.Close();
                try
                {
                    client.updateMarkList(marks.ToArray());
                    dt=client.getMarkDataTable(sID);
                    dv=dt.DefaultView;
                    mGrid.ItemsSource=dv;
                }
                catch(Exception exp) { showMsg(exp.Message); }
            }
        }
    }
```

实际上在批量上传成绩时只用到学生的学号与成绩，学生的姓名是没有用到的，但是保留姓名的信息方便核准学生成绩。

3.4 微信开发平台搭建

3.4.1 案例展示

申请一个公网的 Web 服务器，通过 URL 地址可以访问该服务器。同时申请一个微信公众号，在微信中可以关注该公众号。

3.4.2 技术要点

1. 微信开发平台概述

目前微信是使用最广泛的移动 App 之一，微信不但可以用作人与人之间的聊天工具，用户还能申请公众号，通过公众号与自己的服务器进行通信，实现需要的应用。目前

基于微信公众号的商业应用越来越多,基于微信的开发也越来越流行。

要进行微信开发必须先申请一个公网的服务器,公网服务器也就是能通过 IP 地址或者域名访问的公众 Web 服务器。如果读者已经有了公网的服务器,那就不用再申请,只要提供它的 URL 地址即可。本教材基于应用目的申请一个 Windows 系统的公网的 Web 服务器,该服务器上运行 IIS 与 SQL Server 数据库。为了与微信服务器区分开,不妨把这个服务器称为业务服务器。

除了有一个公网的业务服务器外,还要申请一个微信公众号。微信公众号的类型比较多,有订阅号、服务号等,功能也有点不同。个人申请的公众号一般为订阅号,企业申请的公众号一般为功能强大的服务号。本教材基于教学目的申请了一个微信订阅号。

微信开发平台实际上是由移动端手机、微信服务器、业务服务器 3 个部分组成的。一般手机移动端的功能是通过微信公众号完成人机的交互,数据存储与数据逻辑在业务服务器完成,微信服务器则在这两者之间架起了一个桥梁,完成数据的传输功能。

假设学生成绩存储在业务服务器上,现在要通过手机的微信查询成绩,下面以这个实例来说明一般 App 应用与微信应用的区别。

2. App 应用框架

一般学生要通过手机查询这些成绩,就必须在手机上安装一个查成绩的 App。这个 App 接收学生输入的学号与密码,然后把学号与密码信息发送给业务服务器,服务器接收到学号与密码后到数据库中查找该学生的成绩,然后把结果再返回给这个 App,学生就可以看见结果了。这个 App 的系统结构如图 3-16 所示。

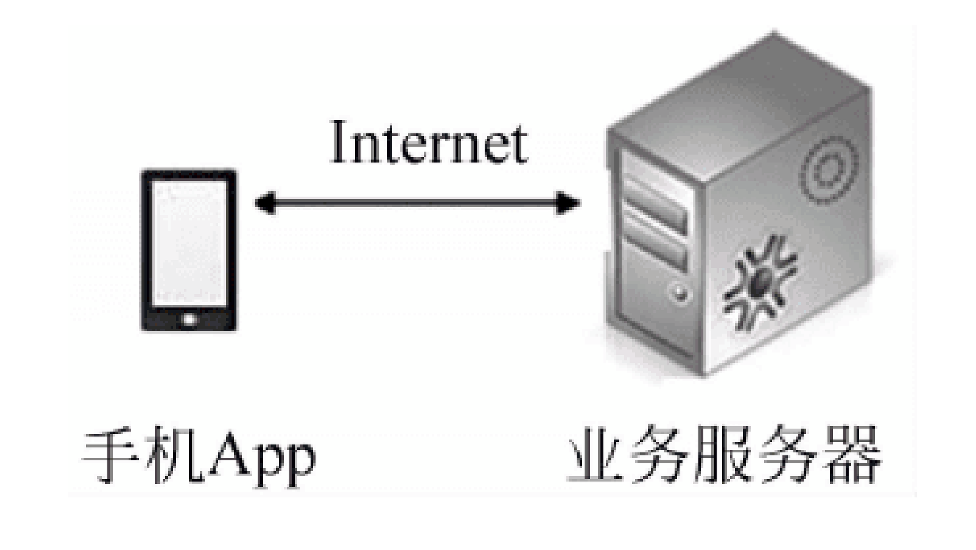


图 3-16 App 应用框架

这种两点式的组合是典型的客户端与服务器程序结构,它的优点是结构简单,速度快。但是它有一个很大缺点是开发成本高,除了开发后台的成绩管理网站外,按目前流行的手机系统至少要开发一套 Android 系统下的 App,还要开发一套 iOS 系统下的 App,用户根据自己的手机类型安装不同版本的 App,才能实现成绩查询。

3. 微信应用框架

基于微信的开发可以借助现有的微信平台降低开发成本,因为微信本身就是一个 App,已经有了 Android 系统与 iOS 系统下的各种版本,而且应用广泛,不需要反复安装。但是微信是与它自己的微信服务器通信的,而微信服务器没有学生的成绩数据,因此需要微信服务器与存储成绩的业务服务器建立通信。学生查询成绩之前可以先申请一个公众号,然后把这个公众号绑定到业务服务器。查询成绩时学生通过微信公众号输入学号与密码信息,微信把学号与密码发给微信服务器,微信服务器进一步把学号与密码推送给业务服务器。最后业务服务器完成成绩查询,把结果再原路返回给微信服务器,微信服务器再返回给学生手机用户,如图 3-17 所示。

这个结构显然是三点式的,最大的优点是不用再开发不同版本的手机 App,微信就是我们的 App。我们主要做的事情就是开发后台的成绩网站,然后遵循微信定义的数据

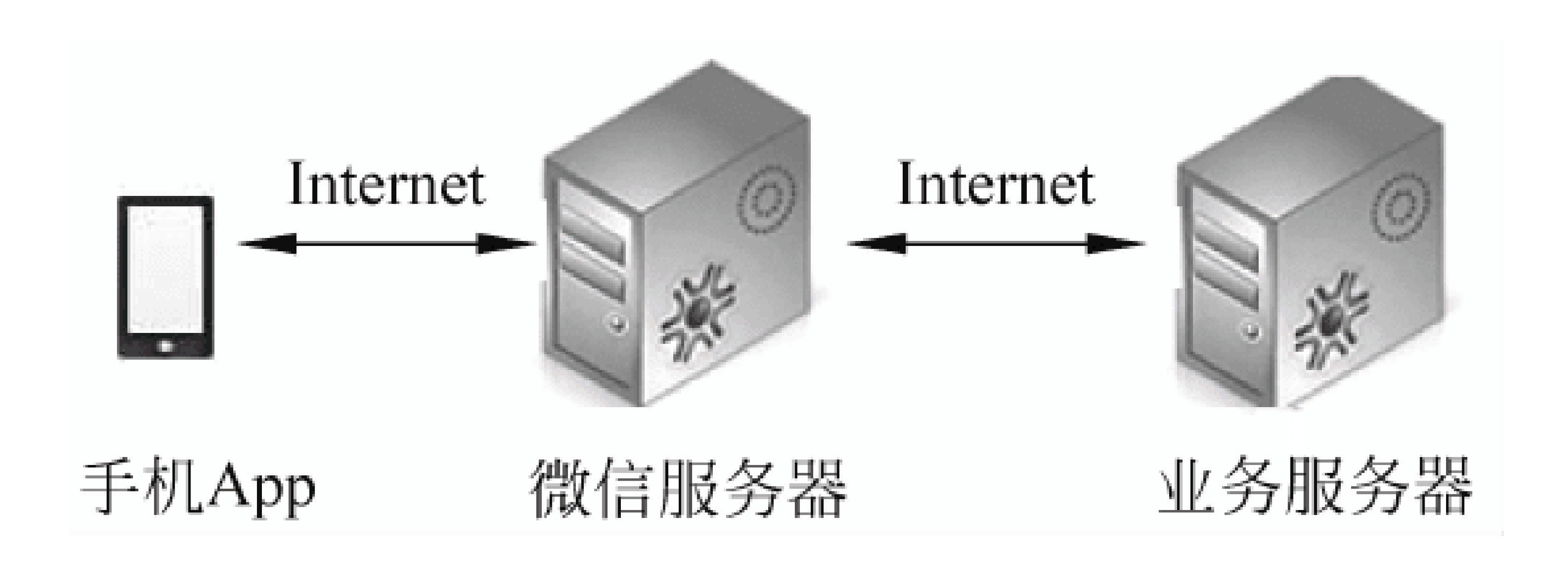


图 3-17 微信应用框架

接口实现数据的通信就可以了。这种模式有极大的优点,而且能满足大部分应用要求。其缺点就是 App 端的表现形式不够多样,已经被微信的格式定死了,速度也不够快,但是对大多数应用来说已经足够了。

3.4.3 服务器申请

1. 申请微信公众号

申请一个微信公众号,以便通过手机等设备能与微信服务器进行通信。进入微信公众平台的网站 https://mp.weixin.qq.com/,按要求填写 E-mail 地址与手机号码等信息,就可以申请微信公众号了。个人申请订阅号是免费的,订阅号在很多功能上受到限制,但是本教程的目的是引导读者进入微信开发的大门,因此订阅号也够用了。

2. 申请业务服务器

从上面的分析可以看到,无论是开发自己的 App 还是借助微信作为 App,都必须有一个公网的 Web 服务器。公网服务器也就是能通过 IP 地址或者域名访问的 Web 服务器。目前提供这种服务器的运营商很多,例如腾讯云、阿里云等。本教程讲解的是微软 .NET 技术的编程技术,因此要求服务器必须是 Windows 服务器,配置 .NET 系统与 IIS 服务器。申请这类服务器一般都需要一定的手续,而且还要支付一定的费用,但是对于学生用户,腾讯云与阿里云都提供了巨大的优惠,学生用户申请一个这样的服务器价格十分低廉。

由于不同的运营商申请过程不大一样,而且随时在改变,读者可以查询运营商的文档说明按要求申请,本书不再赘述。

3.4.4 服务器绑定

1. 绑定 URL 地址

要实现订阅号与业务服务器的通信,就必须绑定该订阅号到业务服务器。绑定的方法是:登录微信公众号平台,进入自己的公众号,单击“开发”中的“基本设置”,如图 3-18 所示。

这个 URL 网址就是你自己申请的业务服务器地址,在没有绑定域名时可以直接填写 IP 地址,如果业务服务器有域名,可以填写域名。这个地址很重要,微信服务器就是通过这个地址与业务服务器进行通信的。

图 3-18 绑定订阅号

Token 是微信服务器要求的一个标签，自己可以随意填写，例如本教程填写 DotNetWeb。EncodingAESKey 是微信服务器的一个安全信息，可以随机生成。

填写完毕如果提交，则会看到一条验证失败的信息。这是为什么呢？原来在提交时微信服务器会根据 URL 地址向业务服务器发送一组验证信息，如果业务服务器没有进行验证并把验证结果返回微信服务器，验证就会失败。

2. 编写验证程序

要完成验证，首先查看验证的规则。进入微信开发者中心，查看微信开发基本设置文档，获悉在服务器验证时微信服务器会向业务服务器发送 signature、timestamp、nonce、echostr 4 个参数。业务服务器必须获取这些参数，并把 token、timestamp、nonce 进行字典排序，然后连接成一个字符串，最后用 SHA1 对它进行哈希计算得到一个哈希字符串。如果这个字符串与微信服务器发来的 signature 一样，那么就确认通过，业务服务器要向微信服务器发回完全一样的 echostr 字符串，微信服务器接收到这个 echostr 后就通过验证。

根据这样的验证要求，编写一个 Default.aspx 程序如下：

```
%@ Page Language="C# " %>
<script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
        String msg="";
        String signature=Request.QueryString["signature"];
        String timestamp=Request.QueryString["timestamp"];
        String nonce=Request.QueryString["nonce"];
        String echostr=Request.QueryString["echostr"];
        if(signature !=null && timestamp !=null && nonce !=null && echostr !=
       null)
        {
            String token="DotNetWeb";
            String[] ArrTmp={ token, timestamp, nonce };
            Array.Sort(ArrTmp);          //字典排序
            string tmpStr=String.Join("", ArrTmp);
```

```
            tmpStr=FormsAuthentication.HashPasswordForStoringInConfigFile
            (tmpStr, "SHA1");
            tmpStr=tmpStr.ToLower();
            if(tmpStr==signature) msg=echostr;
        }
        Response.Write(msg);
    }
</script>
```

3. 部署验证程序

远程登录到业务服务器，把这个程序部署在业务服务器默认的 Web 目录 wwwroot 下，并设置该目录下默认的执行程序首选是 Default. aspx（这是 IIS 默认的，不用再另外设置）。

4. 验证业务服务器

这时再单击微信订阅号基本设置里的“提交”按钮，就可看到验证成功的信息。验证成功后，微信订阅号就与业务服务器绑定起来了，也就是建立了“订阅号—微信服务器—业务服务器”的信息通道，信息就可以在这个通道上来回传输了。

3.4.5 拓展训练

读者可能好奇微信服务器发送的 ignature、timestamp、nonce、echostr 到底是什么，实际上读者可以在验证程序中增加一个日志记录函数 writeLog，把微信服务器推送的信息存储到磁盘文件中，Default. aspx 程序修改如下：

```
<script runat="server">
    void writeLog(String s)
    {
        StreamWriter sw=new StreamWriter(Server.MapPath("log.txt"),true);
        sw.WriteLine(s);
        sw.Close();
    }
    void Page_Load(object sender, EventArgs e)
    {
        String msg="";
        String signature=Request.QueryString["signature"];
        String timestamp=Request.QueryString["timestamp"];
        String nonce=Request.QueryString["nonce"];
        String echostr=Request.QueryString["echostr"];
        if(signature !=null && timestamp !=null && nonce !=null && echostr !=
       null)
        {
```

```
                writeLog(signature+"\r\n"+timestamp+"\r\n"+nonce+"\r\n"+
                echostr);
                String token="DotNetWeb";
                String[] ArrTmp={ token, timestamp, nonce };
                Array.Sort(ArrTmp);      //字典排序
                string tmpStr=String.Join("", ArrTmp);
                tmpStr=FormsAuthentication.HashPasswordForStoringInConfigFile
                (tmpStr, "SHA1");
                tmpStr=tmpStr.ToLower();
                if(tmpStr==signature) msg=echostr;
            }
            Response.Write(msg);
        }
</script>
```

在微信订阅号验证过程中，log.txt 文件会记录微信服务器发送给业务服务器的数据，其中一次记录如下：

```
signature=66d28d179ae432b822c54fe9f1862bc10807fb70
timestamp=1477195000
nonce=773178279
echostr=1195720357539970555
```

由此可见，微信服务器发送给业务服务器的数据主要是一些数字组成的字符串，其中 signature 是经过 timestamp、nonce 与 token 哈希计算得出的字符串。

细心的读者很快会发现，实际上微信服务器就是需要业务服务器返回它发送的 echostr 字符串，如果业务服务器不需要确认是微信服务器的信息，直接把 Default.aspx 简化成如下的形式也能完成验证：

```
<script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
        String echostr=Request.QueryString["echostr"];
        Response.Write(echostr);
    }
</script>
```

3.5 微信事件与文本回复

3.5.1 案例展示

用户通过在微信中搜索公众号进行关注或者扫描该公众号二维码进行关注时，会收到一条“欢迎关注 DotNetWeb 课程”的信息，如图 3-19 所示。



图 3-19　关注微信成绩查询公众号

3.5.2　技术要点

1. 关注事件

如果一个公众号处于开发者模式，当用户关注公众号时，微信服务器就会把一组信息推送给公众号绑定的业务服务器，由该服务器做出相应的处理。按照微信文档的说明，推送的关注事件的信息是一个 XML 字符串，格式如下：

```
<xml>
    <ToUserName><![CDATA[toUser]]></ToUserName>
    <FromUserName><![CDATA[FromUser]]></FromUserName>
    <CreateTime>123456789</CreateTime>
    <MsgType><![CDATA[event]]></MsgType>
    <Event><![CDATA[subscribe]]></Event>
</xml>
```

参数含义如表 3-1 所示。

表 3-1　关注事件参数

参　　数	备　　注
ToUserName	开发者公众微信号，可以看成是公众号的唯一标识，每个公众号都有一个这样的标识
FromUserName	发送用户的账号，可以看成用户的微信号，实际的值是微信号做了处理的公开号(OpenID)，通过它可以唯一确定一个微信用户
CreateTime	消息创建时间，是一个整数值
MsgType	消息类型，对于关注事件，这个值为 event
Event	事件类型，对于关注事件，这个值为 subscribe

微信服务器将这个字符串推送到业务服务器，业务服务器可以先从它的输入数据流中用 BinaryRead 读出二进制数据，然后用 UTF8 编码的 GetString 转为字符串就得到这个 XML 字符串，具体方法如下：

```
byte[] buf=context.Request.BinaryRead(context.Request.TotalBytes);
String xml=Encoding.UTF8.GetString(buf);
```

由于得到的是整个 XML 字符串，要取出每个元素的值还要借助 XML 解析工具，可以使用 XElement 来解析。XElement 的 Parse 函数可以把一个 XML 字符串变成一个 XElement 对象，这是这个 XML 字符串的根对象：

```
XElement root=XElement.Parse(xml);
```

这个对象 root 是根对象<xml>，它下面包含 ToUserName、FromUserName 等元素，通过 Element 得到各个元素及对应的值 Value：

```
String FromUserName=root.Element("FromUserName").Value;
String ToUserName=root.Element("ToUserName").Value;
String CreateTime=root.Element("CreateTime").Value;
String MsgType=root.Element("MsgType").Value;
```

当业务服务器收到这些信息后，就可以根据 FromUseName 得知是谁发来的，然后就可以向该用户回复信息。

2. 回复信息

业务服务器收到关注事件的信息后，可以根据微信的要求按以下 XML 格式向用户回复文本信息：

```
<xml>
<ToUserName><![CDATA[toUser]]></ToUserName>
<FromUserName><![CDATA[fromUser]]></FromUserName>
<CreateTime>12345678</CreateTime>
<MsgType><![CDATA[text]]></MsgType>
<Content><![CDATA[你好]]></Content>
</xml>
```

各个参数含义如表 3-2 所示。

表 3-2 文本信息参数

参 数	备 注
ToUserName	要回复信息的用户，这个用户的名称就是关注事件中的 FromUserName
FromUserName	公众号的名称，也就是关注事件的 ToUserName
CreateTime	一个事件整数值，可以与关注事件的 CreateTime 值一样
MsgType	值为 text，表示该信息是文本信息
Content	要发送的信息内容，可以是任意字符串

如果是直接拼接这个 XML 回复字符串，注意在每个 XML 元素的值中都用<![DATA[元素值]]>来限定，原因是有些值可能包含一些 XML 元素的“<”、“>”等界定符号，这些值如果不使用这个方法来限定，就会影响 XML 的结构。例如，回复的信息是<AB/>，那么<Content>元素变成

```
<Content><AB/></Content>
```

XML 解析器不知道<AB/>到底是<Content>的值还是<Content>包含的一个新元素<AB/>。但是如果写成

```
<Content><![DATA[<AB/>]]></Content>
```

XML 解析器就知道<AB/>是<Content>的值。

3.5.3 接口程序

1. 关注与回复程序

关注事件与回复信息可以设计在 MarkBLL 的项目中，在该项目中加入一个 WeChatManager 类文件，其中 executeCommand 负责用 XElement 解析 XML 字符串，如果发现 MsgType 是一个关注事件，就调用 responseText 函数回复用户一句欢迎词"欢迎关注微信成绩查询公众号！"，设计接口程序如下：

```
namespac MarkBLL
{
    public class WeChatManager
    {
        String responseText(String FromUserName, String ToUserName, String
        CreateTime,String msg)
        {
            String s="<xml>";
            s=s+"<ToUserName><![CDATA["+FromUserName+"]]></ToUserName>";
            s=s+"<FromUserName><![CDATA["+ToUserName+"]]></FromUserName>";
            s=s+"<CreateTime><![CDATA["+CreateTime+"]]></CreateTime>";
            s=s+"<MsgType><![CDATA[text]]></MsgType>";
            s=s+"<Content><![CDATA["+msg+"]]></Content>";
            s=s+"</xml>";
            return s;
        }
        public String executeCommand(String xml)
        {
            String msg="";
            XElement root=XElement.Parse(xml);
            String FromUserName=root.Element("FromUserName").Value;
            String ToUserName=root.Element("ToUserName").Value;
            String CreateTime=root.Element("CreateTime").Value;
            String MsgType=root.Element("MsgType").Value;
            if(MsgType=="event")
            {
              if(root.Element("Event").Value=="subscribe")
```

```
                msg="欢迎关注微信成绩查询公众号！";
            }
            return responseText(FromUserName,ToUserName,CreateTime,msg);
        }
    }
}
```

2. 微信接口程序

设计一个微信接口程序 WeChatApi.ashx 来接收微信服务器的 XML 字符串信息，把这个程序设置为业务服务器的默认处理程序，该程序从输入流中读取二进制数据后转为文本字符串，调用 Manager 中的 executecommand 函数，返回的值是回复的 XML 字符串，直接输出到微信服务器。

关注与回复的详细过程如下：

(1) 用户在微信中关注课程成绩公众号，微信把关注信息发送给微信服务器，这个过程不用我们关心，这是设计微信时就设计好的。

(2) 微信服务器接收到关注信息后，向业务服务器推送一个 XML 文本字符串，这个字符串就上面说到的关注事件字符串，各个参数详见表 3-1。

(3) 业务服务器收到任何 Web 请求时，会执行它默认的响应程序 WeChatApi.ashx，通过 WeChatApi.ashx 程序调用 MarkBLL 中的 executeCommand 函数判断微信服务器推送来的信息是否为关注信息，如果是，就返回一个欢迎词信息。

(4) 业务服务器把欢迎词信息组织成一个新的 XML 文本返回给微信服务器，这个 XML 字符串的参数详见表 3-2。

(5) 微信服务器接收该回复的字符串，根据 FromUserName 信息把欢迎词推送给查询用户。

根据这个过程设计接口程序如下：

```
public class WeChatApi: IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        String msg="";
        try
        {
            byte[] buf=context.Request.BinaryRead(context.Request.TotalBytes);
            if(buf !=null && buf.Length>0)
            {
                WeChatManager manager=new WeChatManager();
                msg=manager.executeCommand(System.Text.Encoding.UTF8.GetString
                (buf));
            }
        }
```

```
            catch { }
            context.Response.Clear();
            context.Response.ContentType="text/plain";
            context.Response.Write(msg);
            context.Response.Flush();
        }
        public bool IsReusable {
            get {
                return false;
            }
        }
    }
```

如果程序正常运行，就会向微信服务器返回一个文本的 XML 字符串，微信服务器解析这个 XML 字符串后会把欢迎词推送给用户。如果出现异常，就回复一个空字符串，微信服务器接收到空字符串时什么都不做。

3.5.4 部署程序

WeChatApi. ashx 设计好后要正确部署到自己的业务服务器中，以便微信服务器能找到它。为此可以把 WeChatApi. ashx 放在业务服务器的默认网站的 wwwroot 目录下，并设置它为首选的默认处理程序，如图 3-20 所示。



图 3-20 部署接口程序

当用户关注这个公众号时，微信服务器就把关注信息推送给业务服务器，业务服务器执行默认的 WeChatApi. ashx 程序，获取关注者的信息，然后回复一句欢迎词给关注者。

因为在设置公众号业务服务器时要执行默认的验证程序 Default. aspx，在用户关注公众号时又要执行 WeChatApi. ashx 程序。因此可以在验证时先把 Default. aspx 设置为第一优先执行的程序，因为验证一次后就不用再验证了，验证完毕再把 WeChatApi. ashx 设置为第一优先执行程序，以后微信服务器有信息推送到业务服务器时，业务服务器就都执行 WeChatApi. ashx 程序了。

3.5.5 拓展训练

在用户关注微信公众号时，微信服务器向业务服务器发送的信息是一个如下的XML字符串：

```
<xml>
<ToUserName><![CDATA[toUser]]></ToUserName>
<FromUserName><![CDATA[FromUser]]></FromUserName>
<CreateTime>123456789</CreateTime>
<MsgType><![CDATA[event]]></MsgType>
<Event><![CDATA[subscribe]]></Event>
</xml>
```

其中FromUserName是用户的公开号OpenID，可以作为识别用户的唯一标识。因此可以在用户关注公众号时记录下这个FromUserName并存储这个号码，就知道什么用户关注了这个公众号。

实际上当用户取消关注这个公众号时，微信服务器也向业务服务器发送类似的信息，只是<Event>的值变成了unsubscribe，因此在这个时候可以知道哪个用户退出了这个公众号。

根据这个关注与取消关注的过程就可以管理关注的用户，基本程序结构如下：

```
if(MsgType=="event")
{
    String eventValue=root.Element("Event").Value;
    if( eventValue=="subscribe")
    {
        msg="欢迎关注微信成绩查询公众号!";
        register(FromUserName);
    }
    elseif( eventValue=="unsubscribe") unregister(FromUserName);
}
```

设计register与unregister函数，其中register(FromUserName)是把FromUserName登记到数据库中，unregister(FromUserName)是从数据库中删除FromUserName。只要查询数据库就知道有哪些用户关注了该公众号。

3.6 微信成绩查询

3.6.1 案例展示

学生用户在公众号中输入自己的学号与密码，就可以查到自己的成绩，在学号与密码不正确时给出错误信息，如图3-21所示。



图 3-21 微信查询成绩

3.6.2 技术要点

1. 接收文本信息

根据微信开发文档的说明，用户在公众号中输入文本信息发送到微信服务器时，微信服务器会把用户输入的文本信息包装成如下格式的 XML 字符串发送给业务服务器：

```
<xml>
  <ToUserName><![CDATA[toUser]]></ToUserName>
  <FromUserName><![CDATA[fromUser]]></FromUserName>
  <CreateTime>1348831860</CreateTime>
  <MsgType><![CDATA[text]]></MsgType>
  <Content><![CDATA[this is a test]]></Content>
  <MsgId>1234567890123456</MsgId>
</xml>
```

其中各个元素的含义与回复文本的类似，<MsgType>的值是 text，<Content>的值是用户输入的文本，<MsgId>是微信服务器给出的信息编号。业务服务器只要得到并解析这个 XML 字符串就知道用户发来的信息。

2. 查询成绩程序

查询成绩的工作在 MarkDAL 与 MarkBLL 中完成。在 MarkDAL 中设计 getStudentMarks 函数，通过学号 sNo 与密码 sPass 就可以从数据库表 students、marks、

subjects 中查询到该学生的各科成绩，查询的 SQL 是一个联合查询命令：

```
select st.sNo as sNo,sName,sSubject,mMark from students st join marks m on st.
sNo=m.sNo join subjects su on m.sID=su.sID and st.sNo=@sNo and st.sPass=
@sPass order by sSubject
```

该命令联合了 students、marks 与 subjects 表，根据学号参数@sNo 与密码参数@sPass 查询学生的姓名、课程、成绩等信息。查询程序设计如下：

```
namespace MarkDAL
{
    public class MarkService
    {
        public String getStudentMarks(String sNo,String sPass)
        {
            String s="";
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                SqlParameter pNo = new SqlParameter { ParameterName ="@sNo",
                SqlDbType=SqlDbType.Char, Value=sNo };
                SqlParameter pPass= new SqlParameter { ParameterName="@sPass",
                SqlDbType=SqlDbType.Char, Value=sPass };
                String sql ="select st.sNo as sNo, sName, sSubject, mMark from
                students st join marks m on st.sNo=m.sNo join subjects su on
                m.sID=su.sID and st.sNo=@sNo and st.sPass=@sPass order by
                sSubject";
                SqlDataReader reader=DB.getReader(sql,pNo,pPass);
                while (reader.Read())
                {
                    if(s=="") s=reader["sNo"].ToString()+
                    "\r\n"+reader["sName"].ToString();
                    s=s+"\r\n"+reader["sSubject"].ToString()+
                    ": "+reader["mMark"].ToString();
                }
                reader.Close();
                con.Close();
            }
            return s;
        }
    }
}
```

输出的结果是一个文本字符串，其中学号、姓名、各门课程的成绩各占一行，这样的格式方便在微信中显示。

然后在 MarkBLL 中设计一个 getStudentMarks 函数,用来调用 MarkDAL 的对应函数,这个函数如下:

```
namespace MarkBLL
{
    public class MarkManager
    {
        public String getStudentMarks(String sNo,String sPass)
        {
            MarkService service=new MarkService();
            return service.getStudentMarks(sNo,sPass);
        }
    }
}
```

3. 成绩查询命令

用户发送的文本都集中在<Content>的字符串中,必须约定一种格式来区分哪个部分是学号,哪个部分是密码,即要定义一个简单的协议。如果学号与密码都是比较简单的字符串,不包含逗号等特殊的字符,那么可以约定用逗号来分开它们,即命令格式是

```
学号,密码
```

协议就是双方的一种约定,一旦这样规定,业务服务器在解析学号与密码时也按这样的原则进行,即找到它们之间的逗号,把这个命令字符串拆分成两个部分,前半部分就是学号,后半部分就是密码,就可以进行成绩查询了。

3.6.3 接口程序

成绩查询过程如下:

(1) 用户在公众号中按命令格式输入以下文本:

```
学号,密码
```

把学号与密码信息发送给微信服务器。

(2) 微信服务器接收到学号与密码后,把它打包成一个文本的 XML 字符串推送给业务服务器。

(3) 业务服务器通过 WeChatApi.ashx 程序调用 MarkBLL 中的 executeCommand 函数判断微信服务器推送来的信息是否为文本信息,如果是,就分解出学号与密码,然后调用 MarkDAL 的 getStudentMarks 函数获取成绩作为返回信息。

(4) 业务服务器把返回的成绩信息组织成一个新的 XML 文本返回给微信服务器。

(5) 微信服务器根据 FromUserName 信息把成绩信息推送给查询用户,完成查询过程。

根据前面的分析,接口程序设计如下:

```
namespace MarkBLL
{
    public class WeChatManager
    {
        String responseText (String FromUserName, String ToUserName, String
        CreateTime,String msg)
        {
            String s="<xml>";
            s=s+"<ToUserName><![CDATA["+FromUserName+"]]></ToUserName>";
            s=s+"<FromUserName><![CDATA ["+ToUserName+"]]></FromUserName>";
            s=s+"<CreateTime><![CDATA["+CreateTime+"]]></CreateTime>";
            s=s+"<MsgType><![CDATA[text]]></MsgType>";
            s=s+"<Content><![CDATA["+msg+"]]></Content>";
            s=s+"</xml>";
            return s;
        }
        public String executeCommand(String xml)
        {
            String msg="";
            XElement root=XElement.Parse(xml);
            String FromUserName=root.Element("FromUserName").Value;
            String ToUserName=root.Element("ToUserName").Value;
            String CreateTime=root.Element("CreateTime").Value;
            String MsgType=root.Element("MsgType").Value;
            try
            {
                if(MsgType=="text")
                {
                    String content = root.Element ("Content").Value.Trim().
                    ToLower();
                    String[] st=content.Split(new char[] { ',' },
                    StringSplitOptions.RemoveEmptyEntries);
                    if(st.Length==2)
                    {
                        MarkManager manager=new MarkManager();
                        msg=manager.getStudentMarks(st[0], st[1]);
                        if(msg=="") msg="查无成绩,是不是学号或者密码输入错误?";
                    }
                    else  msg="查询成绩:学号,密码";
                }
            }
            catch(Exception exp) { msg=exp.Message; }
            return responseText(FromUserName,ToUserName,CreateTime,msg);
        }
```

```
        }
    }
```

3.6.4 部署程序

WeChatApi.ashx 也部署在业务服务器的 wwwroot 默认 web 目录中，并设置它为业务服务器默认处理程序的首选。

3.6.5 拓展训练

1. 修改密码程序

类似于成绩查询功能，还可以设计用户修改密码的功能。在 MarkDAL 中增加一个修改密码函数 changePassword，它有学号 sNo、旧密码 oldPass 与新密码 newPass 共 3 个参数，当通过学号与旧密码能在学生表 students 中找到一条记录时，就把密码修改为新密码，程序如下：

```
namespace MarkDAL
{
    public class StudentService
    {
        public bool changePassword(String sNo,String oldPass,String newPass)
        {
            bool flag=false;
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                SqlParameter pNo = new SqlParameter { ParameterName = "@sNo",
                SqlDbType=SqlDbType.Char, Value=sNo };
                SqlParameter pOldPass=new SqlParameter { ParameterName=
                "@oldPass", SqlDbType=SqlDbType.Char, Value=oldPass };
                SqlParameter pNewPass=new SqlParameter { ParameterName=
                "@newPass", SqlDbType=SqlDbType.Char, Value=newPass };
                if(DB.executeCommand("update students set sPass = @ newPass
                where sNo=@sNo and sPass=@oldPass", pNo, pNewPass, pOldPass)
                >0) flag=true;
                con.Close();
            }
            return flag;
        }
    }
}
```



2. 修改密码命令

用户发送的文本都集中在<Content>的字符串中，按照前面类似的约定，如果用户输入的命令格式是

学号,旧密码,新密码

业务服务器收到的字符串是用逗号分开的 3 个部分，就是修改密码命令；如果收到的是两个部分，就是查询成绩命令。

3. 设计接口程序

适当修改 executeCommand 函数，增加修改密码的功能：

```
public String executeCommand(String xml)
{
    String msg="";
    XElement root=XElement.Parse(xml);
    String FromUserName=root.Element("FromUserName").Value;
    String ToUserName=root.Element("ToUserName").Value;
    String CreateTime=root.Element("CreateTime").Value;
    String MsgType=root.Element("MsgType").Value;
    try
    {
        if(MsgType=="event")
        {
            String eventValue=root.Element("Event").Value;
            if( eventValue=="subscribe") msg="欢迎关注微信成绩查询公众号!";
        }
        else if(MsgType=="text")
        {
            String content=root.Element("Content").Value.Trim().ToLower();
            String[] st=content.Split(new char[] { ',' }, StringSplitOptions.
            RemoveEmptyEntries);
            if(st.Length==2)
            {
                MarkManager manager=new MarkManager();
                msg=manager.getStudentMarks(st[0], st[1]);
                if(msg=="") msg="查无成绩,是不是学号或者密码输入错误?";
            }
            else if(st.Length==3)
            {
                StudentManager manager=new StudentManager();
                if(manager.changePassword(st[0], st[1], st[2])) msg="密码修改
                成功!";
```

```
                    else msg="密码修改失败!";
                }
                else  msg="查询成绩:学号,密码\r\n 修改密码:学号,旧密码,新密码";
            }
        }
        catch(Exception exp) { msg=exp.Message; }
        return responseText(FromUserName,ToUserName,CreateTime,msg);
    }
```

在公众号中输入修改密码的命令就可以修改密码,如图 3-22 所示。



图 3-22 微信修改密码

3.7 微信成绩查询程序的实现

3.7.1 技术要点

1. 保护 Web Service 接口函数

在前面的 Web Service 的接口函数程序中都没有对这些函数进行保护,很显然只要这个 Web Service 部署在网站上,任何人都可以找到这个服务,而且可以生成接口函数的调用代理来调用这些函数,因此用户就可以写一个程序来随意改变课程、学生、成绩等记录。这样显然是极为不安全的,必须对程序进行保护。

一个简单的保护方法是在 MarkService. asmx 程序中使用 SoapHeader 类,顾名思义这个类就是 Soap 的头信息类,包含了很多 Soap 信息。程序可以在它的基础上再派生一个类,例如起名为 UserHeader,在类中另外增加两个属性,一个是用户名称 uName,另一

个是用户密码 uPass，这个类定义如下：

```
public class UserHeader : SoapHeader
{
    public String uName;
    public String uPass;
    public bool valid()
    {
        UserManager manager=new UserManager();
        return(manager.login(uName, uPass)=="logined");
    }
}
```

在类中定义了一个检验函数 valid，它的作用是检验这个用户是否有效。login 是 MarkBLL 中定义的用户注册登录程序，这个程序与前面项目中的用户注册登录程序类似。

在定义 UserHeader 后，重新修改 MarkService 类的程序，在类中定义一个 UserHeader的变量 header，同时为每个接口函数前面增加一个[SoapHeader("header")]的属性。例如修改后的增加课程函数 addSubject 如下：

```
public class MarkService: System.Web.Services.WebService
{
    public UserHeader header;
    [SoapHeader("header")]
    [WebMethod]
    public int addSubject(SubjectClass s)
    {
        if(header !=null && header.valid())
        {
            SubjectManager manager=new SubjectManager();
            return manager.addSubject(ref s);
        }
        return 0;
    }
}
```

MarkService.asmx 经过修改后重新部署在网站上，客户端程序重新更新服务引用 WS，那么在客户端会生成 WS.UserHeader 代理类，函数 addSubject 的代理函数变成如下形式：

```
bool addSubject(WS.UserHeader header,WS.SubjectClass subject)
```

就是说在调用 addSubject 函数时要求提供一个 WS.UserHeader 对象，这个对象包含用户名称 uName 与密码 uPass 属性，通过代理函数这个 header 传递给服务器，在调用

服务器函数 addSubject 中可以得到一个 header 对象，在函数中检测 header 对象，如果不为 null，同时用户身份检测通过，就调用增加课程的函数，否则就不执行增加课程的工作。

这个方法有效地保护了 Web Service 的安全性，即便用户能够获取 Web Service 的函数调用规则，每个函数都必须提供 UserHeader 对象进行用户身份检验，一般用户也不能成功调用这些函数了。

2. 回复图文信息

用户在关注公众号与查询成绩时，业务服务器都是回复一个纯文本信息给用户。如果要美观，还可以发送图文信息给用户，图文信息的基本格式如下：

```
<xml>
<ToUserName><![CDATA[toUser]]></ToUserName>
<FromUserName><![CDATA[fromUser]]></FromUserName>
<CreateTime>12345678</CreateTime>
<MsgType><![CDATA[news]]></MsgType>
<ArticleCount>2</ArticleCount>
<Articles>
<item>
<Title><![CDATA[title1]]></Title>
<Description><![CDATA[description1]]></Description>
<PicUrl><![CDATA[picurl]]></PicUrl>
<Url><![CDATA[url]]></Url>
</item>
<item>
<Title><![CDATA[title]]></Title>
<Description><![CDATA[description]]></Description>
<PicUrl><![CDATA[picurl]]></PicUrl>
<Url><![CDATA[url]]></Url>
</item>
</Articles>
</xml>
```

其中可以包含多个图文信息，图文信息的图像由 PicUrl 指定，Description 实际上是摘要，只能是纯文本，Url 是要转去的 Url 原文地址，各个参数的意义如表 3-3 所示。

表 3-3 图文信息参数意义

参 数	意 义
ToUserName	开发者公众微信号，可以看成是公众号的唯一标识
FromUserName	发送用户的账号，可以看成用户的微信号，实际值是微信号做了处理的公开号(OpenID)，通过它可以唯一确定一个微信用户
CreateTime	消息创建时间，是一个整数值

续表

参　　数	意　　义
MsgType	消息类型,对于图文消息固定为 news
ArticleCount	图文的个数,最多 10 个
Articles	一组<item>的图文项目
Title	每个图文的标题
Description	图文的文字描述
PicUrl	图像的 URL 地址,第一个的图像是最大的,其余的为小图
Url	源文的地址,可以为空

根据图文信息的要求,设计回复图文的函数 responseImageText 来代替前面的 responseText 函数。

```
String responseImageText (String FromUserName, String ToUserName, String
CreateTime, String msg)
{
    String s="<xml>";
    s=s+"<ToUserName><![CDATA["+FromUserName+"]]></ToUserName>";
    s=s+"<FromUserName><![CDATA["+ToUserName+"]]></FromUserName>";
    s=s+"<CreateTime><![CDATA["+CreateTime+"]]></CreateTime>";
    s=s+"<MsgType><![CDATA[news]]></MsgType>";
    s=s+"<ArticleCount>1</ArticleCount>";
    s=s+"<Articles><item>";
    s=s+"<Title>微信成绩查询</Title>";
    s=s+"<Description><![CDATA["+msg+"]]></Description>";
    s=s+"<PicUrl><![CDATA[http://119.29.202.190/WeChatMark/mark.jpg]]>
    </PicUrl>";
    s=s+"</item></Articles>";
    s=s+"</xml>";
    return s;
}
```

在回复时只回复一条图文信息,因此<ArticleCount>值为 1,成绩信息放在<Description>中显示,所用的图片<PicUrl>可以是业务服务器上的一张图片,也可以是网络上任何一张图片,但是要详细给出图片的 URL 地址。

3.7.2 服务器程序

服务器管理程序采用 3 层结构,分别定义 MarkDAL、MarkBLL、MarkModel 3 个类项目与一个 web 网站,如图 3-23 所示。

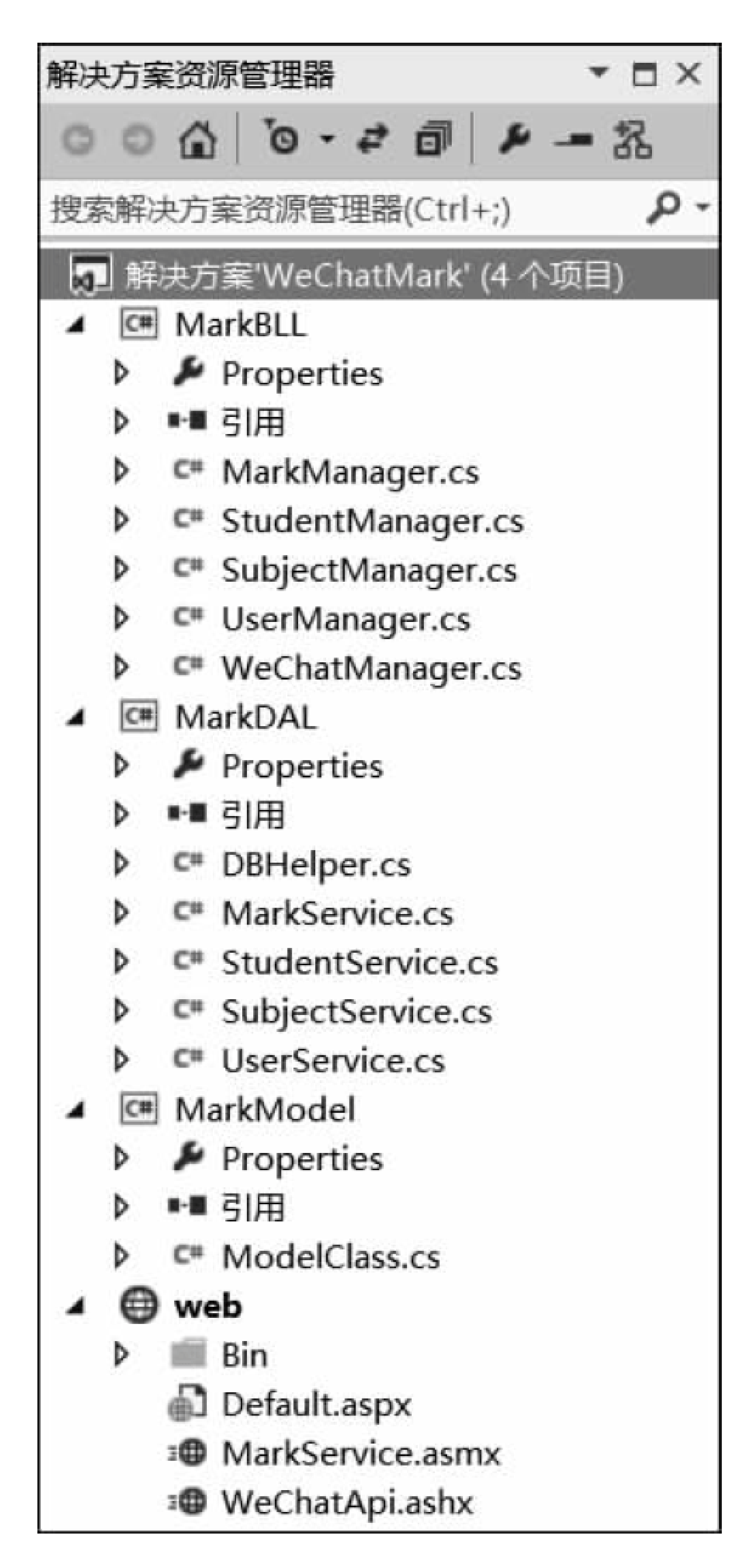


图 3-23 服务器管理程序结构

1. MarkModel

定义 SubjectClass、StudentClass、MarkClass 类，与数据库表 subjects、students、marks 对应。

2. MarkDAL

MarkDAL 包含 DBHlep、SubjectService、StudentService 以及 MarkService 类，各个类主要函数如表 3-4 所示。

表 3-4 MarkDAL 中类余函数

类	函数名称	说明
SubjectService	getSubjectDataTable()	获取课程记录数据集
	addSubject(SubjectClass)	增加一门课程
	deleteSubject(SubjectClass)	删除一门课程
	updateSubject(SubjectClass)	更新一门课程

续表

类	函数名称	说明
StudentService	getStudentDataTable()	获取学生记录数据集
	addStudent(StudentClass)	增加一个学生
	deleteStudent(StudentClass)	删除一个学生
	updateStudent(StudentClass)	更新一个学生
	appendStudentList(List<StudentClass>)	增加一批学生
	changePassword(String,String,String)	修改学生密码
MarkService	getMarkDataTable(int)	获取成绩记录数据集
	updateMarkList(List<MarkClass>)	更新一批成绩
	getStudentMarks(String,String)	微信获取学生成绩
UserService	login(String,String)	用户登录

3. MarkBLL

这个类项目中包含对应的 SubjectManager、StudentManager、MarkManager、UserManager 类，每个类的函数与 MarkDAL 中的对应。另外的 WeChatManager 是用来响应微信服务器的管理类，程序如下：

```
namespace MarkBLL
{
    public class WeChatManager
    {
        String responseImageText (String FromUserName, String ToUserName,
        String CreateTime, String msg)
        {
            String s="<xml>";
            s=s+"<ToUserName><![CDATA["+FromUserName+"]]></ToUserName>";
            s=s+"<FromUserName><![CDATA ["+ToUserName+"]]></FromUserName>";
            s=s+"<CreateTime><![CDATA["+CreateTime+"]]></CreateTime>";
            s=s+"<MsgType><![CDATA[news]]></MsgType>";
            s=s+"<ArticleCount>1</ArticleCount>";
            s=s+"<Articles><item>";
            s=s+"<Title>微信成绩查询</Title>";
            s=s+"<Description><![CDATA["+msg+"]]></Description>";
            s=s+"<PicUrl><![CDATA[http://119.29.202.190/WeChatMark/mark.
            jpg]]></PicUrl>";
            s=s+"</item></Articles>";
            s=s+"</xml>";
            return s;
```

```
        }
        public String executeCommand(String xml)
        {
            String msg="";
            XElement root=XElement.Parse(xml);
            String FromUserName=root.Element("FromUserName").Value;
            String ToUserName=root.Element("ToUserName").Value;
            String CreateTime=root.Element("CreateTime").Value;
            String MsgType=root.Element("MsgType").Value;
            try
            {
                if(MsgType=="event")
                {
                    String eventValue=root.Element("Event").Value;
                    if(eventValue=="subscribe") msg="欢迎关注微信成绩查询公
                    众号!";
                }
                else if(MsgType=="text")
                {
                    String content=root.Element("Content").Value.Trim().
                    ToLower();
                    String[] st=content.Split(new char[] { ',' },
                    StringSplitOptions.RemoveEmptyEntries);
                    if(st.Length==2)
                    {
                        MarkManager manager=new MarkManager();
                        msg=manager.getStudentMarks(st[0], st[1]);
                        if(msg=="") msg="查无成绩,是不是学号或者密码输入错误?";
                    }
                    else if(st.Length==3)
                    {
                        StudentManager manager=new StudentManager();
                        if(manager.changePassword(st[0], st[1], st[2])) msg=
                       "密码修改成功!";
                        else msg="密码修改失败!";
                    }
                    else msg="查询成绩:学号,密码\r\n 修改密码:学号,旧密码,新密码";
                }
            }
            catch(Exception exp) { msg=exp.Message; }
            return responseImageText (FromUserName, ToUserName, CreateTime, msg);
        }
    }
}
```

4. web 网站

web 网站主要包含的程序如表 3-5 所示。

表 3-5　web 网站文件

文　　件	主要功能或者包含函数
Deafult. aspx	微信验证程序
WeChatApi. ashx	微信服务器与业务服务器的接口程序
MarkService. asmx	课程、学生、成绩管理的 Web Service

Default. aspx 与 WeChatApi. ashx 在前面已经讲解过，MarkService. asmx 需要对函数进行保护，程序如下：

```
<%@WebService Language="C# " Class="MarkService" %>
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Data;
using System.Collections.Generic;
using MarkModel;
using MarkBLL;
[WebService(Namespace="http://tempuri.org/")]
[WebServiceBinding(ConformsTo=WsiProfiles.BasicProfile1_1)]
public class UserHeader : SoapHeader
{
    public String uName;
    public String uPass;
    public bool valid()
    {
        UserManager manager=new UserManager();
        return  (manager.login(uName, uPass)=="logined");
    }
}
public class MarkService : System.Web.Services.WebService
{
    public UserHeader header;
    [SoapHeader("header")]
    [WebMethod]
    public bool login()
    {
        return(header !=null && header.valid());
    }
```

```
[SoapHeader("header")]
[WebMethod]
public DataTable getSubjectDataTable()
{
    if(header !=null && header.valid())
    {
        SubjectManager manager=new SubjectManager();
        return manager.getSubjectDataTable();
    }
    return null;
}
[SoapHeader("header")]
[WebMethod]
public bool updateSubject(SubjectClass s)
{
    if(header !=null && header.valid())
    {
        SubjectManager manager=new SubjectManager();
        return manager.updateSubject(s);
    }
    return false;
}
[SoapHeader("header")]
[WebMethod]
public int addSubject(SubjectClass s)
{
    if(header !=null && header.valid())
    {
        SubjectManager manager=new SubjectManager();
        return manager.addSubject(ref s);
    }
    return 0;
}
[SoapHeader("header")]
[WebMethod]
public bool deleteSubject(SubjectClass s)
{
    if(header !=null && header.valid())
    {
        SubjectManager manager=new SubjectManager();
        return manager.deleteSubject(s);
    }
    return false;
}
```

```
//////////////
[SoapHeader("header")]
[WebMethod]
public DataTable getStudentDataTable()
{
    if(header !=null && header.valid())
    {
        StudentManager manager=new StudentManager();
        return manager.getStudentDataTable();
    }
    return null;
}
[SoapHeader("header")]
[WebMethod]
public bool updateStudent(StudentClass s)
{
    if(header !=null && header.valid())
    {
        StudentManager manager=new StudentManager();
        return manager.updateStudent(s);
    }
    return false;
}
[SoapHeader("header")]
[WebMethod]
public bool addStudent(StudentClass s)
{
    if(header !=null && header.valid())
    {
        StudentManager manager=new StudentManager();
        return manager.addStudent(s);
    }
    return false;
}
[SoapHeader("header")]
[WebMethod]
public int appendStudentList(List<StudentClass>students)
{
    if(header !=null && header.valid())
    {
        StudentManager manager=new StudentManager();
        return manager.appendStudentList(students);
    }
    return 0;
```

```
        }
        [SoapHeader("header")]
        [WebMethod]
        public bool deleteStudent(StudentClass s)
        {
            if(header !=null && header.valid())
            {
                StudentManager manager=new StudentManager();
                return manager.deleteStudent(s);
            }
            return false;
        }
        //////////////
        [SoapHeader("header")]
        [WebMethod]
        public DataTable getMarkDataTable(int sID)
        {
            if(header !=null && header.valid())
            {
                MarkManager manager=new MarkManager();
                return manager.getMarkDataTable(sID);
            }
            return null;
        }
        [SoapHeader("header")]
        [WebMethod]
        public int updateMarkList(List<MarkClass>marks)
        {
            if(header !=null && header.valid())
            {
                MarkManager manager=new MarkManager();
                return manager.updateMarkList(marks);
            }
            return 0;
        }
    }
```

服务器程序设计好后，执行“生成解决方案”命令，在网站 web 的 bin 目录下自动生成 MarkDAL、MarkBLL 及 MarkModel 的动态库文件，把它们复制到远程服务器的默认目录 wwwroot 的 bin 目录下，同时把 Default. aspx、WeChatApi. ashx、MarkService. asmx 复制到 wwwroot 目录下，另外把成绩数据库部署在远程服务器中。这些都做好后，就可以通过成绩管理客户端管理成绩，通过微信公众号查询成绩了。

3.7.3 客户端管理程序

客户端是一个多窗体的 WPF 程序，程序结构如图 3-24 所示，它有一个主窗体 MainWindow，用来实现管理员用户的登录，同时用来启动课程管理窗体 SubjectWindow、学生管理窗体 StudentWindow、成绩管理窗体 MarkWindow。



图 3-24 客户端程序结构

MainWindow 的主要功能是实现管理员的登录，登录成功后可以启动课程、学生、成绩的管理窗体，MainWindow 的界面包含的控件如表 3-6 所示。

表 3-6 MainWindow 的控件

控件名称	控件类型	说明
txtName	TextBox	管理员名称
txtPass	TextBox	管理员密码
txtURL	TextBox	服务器地址
btLogin	Button	登录按钮
btSubject	Button	启动课程管理窗体按钮
btStudent	Button	启动学生管理窗体按钮
btMark	Button	启动成绩管理窗体按钮

主要程序如下：

```
namespace WeChatMarkClient
{
```

```
public partial class MainWindow : Window
{
    WS.WeChatMarkServiceSoapClient client;
    WS.UserHeader header;
    public MainWindow()
    {
        InitializeComponent();
    }
    public void showMsg(String s)
    {
        MessageBox.Show(s, "Information", MessageBoxButton.OK);
    }
    public bool confirm(String s)
    {
        return(MessageBox.Show(s, "Confirmation", MessageBoxButton.
        YesNo)==MessageBoxResult.Yes);
    }
    String encryptString(String s)
    {
        MD5 md5=new MD5CryptoServiceProvider();
        byte[] buf=Encoding.UTF8.GetBytes(s);
        buf=md5.ComputeHash(buf);
        s="";
        foreach (byte x in buf) s=s+x.ToString("X2");
        return s;
    }
    private void btLogin_Click(object sender, RoutedEventArgs e)
    {
        String url=txtURL.Text.Trim();
        String uName=txtName.Text.Trim();
        String uPass=txtPass.Password;
        if(url!="" && uName!="" && uPass!="")
        {
            try
            {
                uPass=encryptString(uPass);
                header=new WS.UserHeader { uName=uName, uPass=uPass };
                client=new WS.WeChatMarkServiceSoapClient();
                client.Endpoint.Address=new System.ServiceModel.
                EndpointAddress(url);
                if(client.login(header))
                {
                    btLogin.IsEnabled=false;
```

```
                    txtPass.IsEnabled=false;
                    txtURL.IsEnabled=false;
                    btSubject.IsEnabled=true;
                    btStudent.IsEnabled=true;
                    btMark.IsEnabled=true;
                }
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
    private void mainWindow_Loaded(object sender, RoutedEventArgs e)
    {
        btSubject.IsEnabled=false;
        btStudent.IsEnabled=false;
        btMark.IsEnabled=false;
    }
    private void btSubject_Click(object sender, RoutedEventArgs e)
    {
        SubjectWindow dlg=new SubjectWindow(this, header,client);
        dlg.Show();
        this.Hide();
    }
    private void btStudent_Click(object sender, RoutedEventArgs e)
    {
        StudentWindow dlg=new StudentWindow(this,header, client);
        dlg.Show();
        this.Hide();
    }
    private void btMark_Click(object sender, RoutedEventArgs e)
    {
        MarkWindow dlg=new MarkWindow(this,header, client);
        dlg.Show();
        this.Hide();
    }
  }
}
```

SubjectWindow、StudentWindow、MarkWindow 的界面与程序在前面的章节中基本都讲解过，整合到这个综合项目中只要做一些修改即可，限于篇幅不再列出。

3.7.4 拓展训练

如果业务服务器是域名绑定的，那么用户可以直接从微信公众号转去业务服务器上的任何一个网页。因此可以设计一个简单的成绩查询网页 seek. aspx，该网页让用户在

界面中输入学号与密码，然后单击查询按钮就可以查询成绩了，这与普通的计算机网页查询没有什么两样。seek.aspx 的设计也与普通的计算机网页没有太大区别，唯一不同的是如果直接用手机去浏览计算机网页，一般网页的字体会很小。为了让手机能比较好地浏览计算机网页，网页中要加上一段开始的<head>部分：

```
<head>
<meta name="viewport" content="width=device-width,height=device-height,
inital-scale=1.0,maximum-scale=1.0,user-scalable=no;">
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<meta name="format-detection" content="telephone=no">
</head>
```

如果网页包含这个部分，用手机浏览网页时字体就不会太小，同时又不影响计算机浏览该网页。考虑这个因素后 seek.aspx 程序设计如下：

```
<%@Page Language="C# " Debug="true" Async="true" %>
<%@Import NameSpace="System" %>
<%@Import NameSpace="System.Configuration" %>
<%@Import NameSpace="System.Web" %>
<%@Import Namespace="MarkBLL" %>
<script src="jquery.js"></script>
<script>
   function trySeek() {
      var no=$ ("# txtNo").val();
      var pass=$ ("# txtPass").val();
      if(no=="" || pass=="") {
            alert("学号与密码不能为空"); return false;
      }
      return true;
   }
</script>
<script runat="server">
  void seek(Object sender,EventArgs e)
  {
     try
     {
        String sNo=txtNo.Text.Trim();
        String sPass=txtPass.Text.Trim();
        MarkManager manager=new MarkManager();
        msg.Text=manager.getStudentMarks(sNo,sPass);
     }
     catch(Exception exp) { msg.Text=exp.Message; }
  }
</script>
```

```
<html>
<head>
<meta name="viewport" content="width=device-width,height=device-height,
inital-scale=1.0,maximum-scale=1.0,user-scalable=no;">
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<meta name="format-detection" content="telephone=no">
</head>
<body>
<form id="myform" runat="server" method="post">
    <table width="240" align="center">
        <tr><td>学号</td><td><asp:TextBox id="txtNo" runat="server"
        Text=""  Width="100%" /></td></tr>
        <tr><td>密码</td><td><asp:TextBox id="txtPass" TextMode=
        "Password" runat="server" Text="123" Width="100%"/></td></tr>
        <tr><td colspan="2" align="center"><asp:Button id="btSeek" runat=
        "server" Text="查询" onclick="seek" onClientClick="trySeek()" /></td>
        </tr>
        <tr><td colspan="2" align="center"><asp:TextBox id="msg" runat=
        "server" Text="" TextMode="MultiLine" Rows="10" Columns="30"
        ReadOnly="true"  /></td></tr>
    </table>
</form>
</body>
</html>
```

通过手机直接访问这个网页查询的结果如图 3-25 所示。



图 3-25　手机网页查询成绩

练 习 三

1. 编写一个 Web Service 程序 server. asmx 与一个客户端程序 client，server. asmx 中包含如下接口函数 test：

```
[WebMethod]
public String test()
{
    throw new Exception("Something is wrong");
}
```

编写客户端的程序调用这个函数并捕捉这个异常。

2. 一个城市信息表文本文件是 cities. txt，结构如下：

```
广东，广州
广东，深圳
广东，珠海
四川，成都
四川，广安
贵州，贵阳
…
```

每一行是一个城市，包括城市所在省份、城市名称。编写服务器 Web Service，它能返回所有省份的字符串列表，能根据指定的省份返回该省份下辖的所有城市的字符串列表。编写客户端用两个下拉列表框存储省份与城市，获取所有的省份显示在一个下拉列表框中，在选择一个省份后获取该省份的城市显示在另外一个下拉列表框中。

3. Web 服务器文件夹下有一个 images 的子文件夹，images 中存储了一组 jpg 的图像文件，编写 Web Service 服务器程序与客户端程序实现以下功能：

(1) 获取 images 中所有图像名称的列表，显示在客户端。

(2) 客户端选择一个图像名称后，能查看到该图像。

(3) 客户端能上传图像到服务器的 images 文件夹中，如果有同名的文件存在，则拒绝上传。

4. 编写一个手机微信查询城市天气预报的程序，在微信中输入一个城市的名称后能查到该城市的天气预报。微信接口的服务器程序的功能是接收微信服务器传递过来的城市名称，然后从百度服务器获取天气预报数据，再把天气预报信息推送给微信服务器和手机微信。

项目四



基于 WCF 的试题练习程序

WCF(Windows Communication Foundation)是由微软公司开发的一系列支持数据通信的应用程序框架，可以翻译为 Windows 通信开发平台，它是对 Web Service 的扩展，其最大的特点是服务器可以是一个独立于 IIS 的应用程序，而且支持 HTTP、TCP 等多种协议，在通信安全性上有很强的功能。

试题练习程序是一个客户端与服务器的 WCF 程序系统，程序结构如图 4-1 所示，服务器与客户端通过 SOAP 协议进行交互。

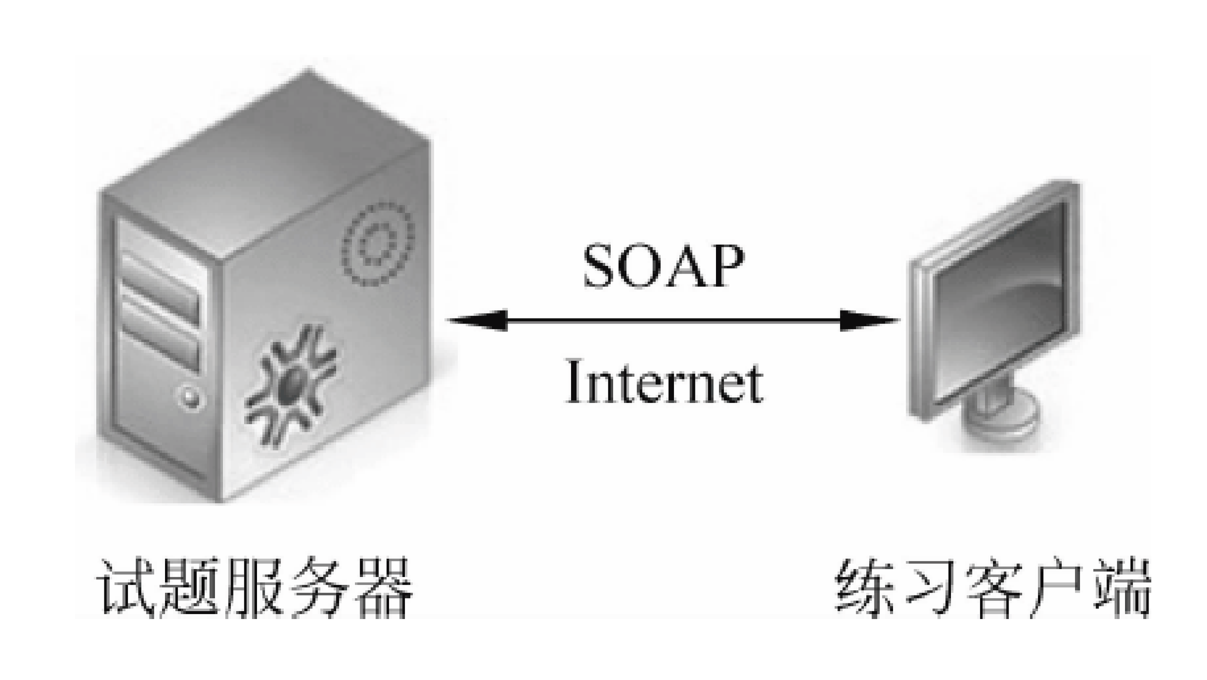


图 4-1　程序结构

试题服务器是一个控制台程序，启动后会监听某个网址，如图 4-2 所示。试题练习客户端程序是一个 WPF 的窗体程序，启动后输入用户名称与密码，单击"登录"按钮可以登录到服务器。其中名为 Admin 的用户是管理员用户，其他名称的为普通用户，如图 4-3 所示。

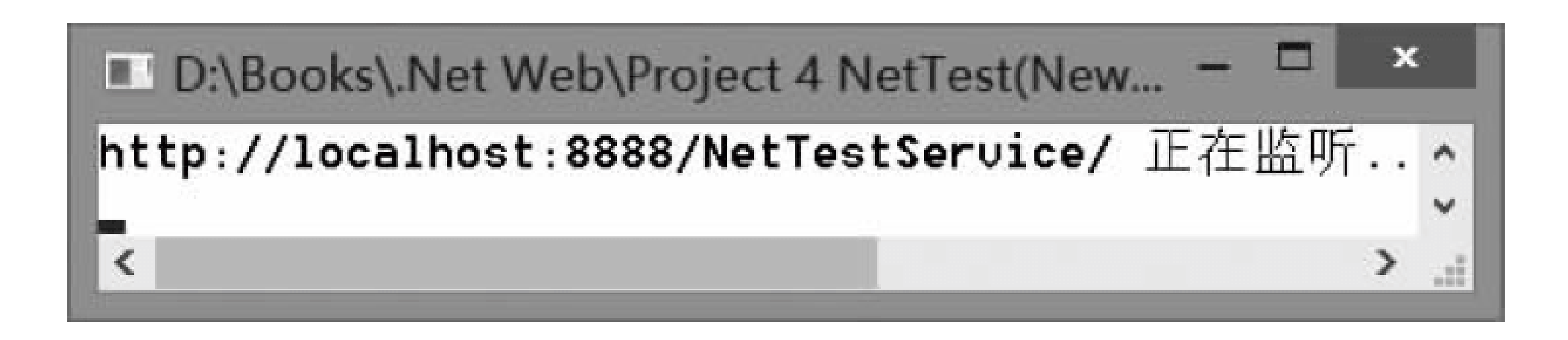


图 4-2　服务器程序

图 4-3　用户登录

管理员的责任是试题与成绩的管理，登录后可以进行试题管理、成绩管理操作，管理员可以增加、删除与修改试题，如图 4-4 所示。

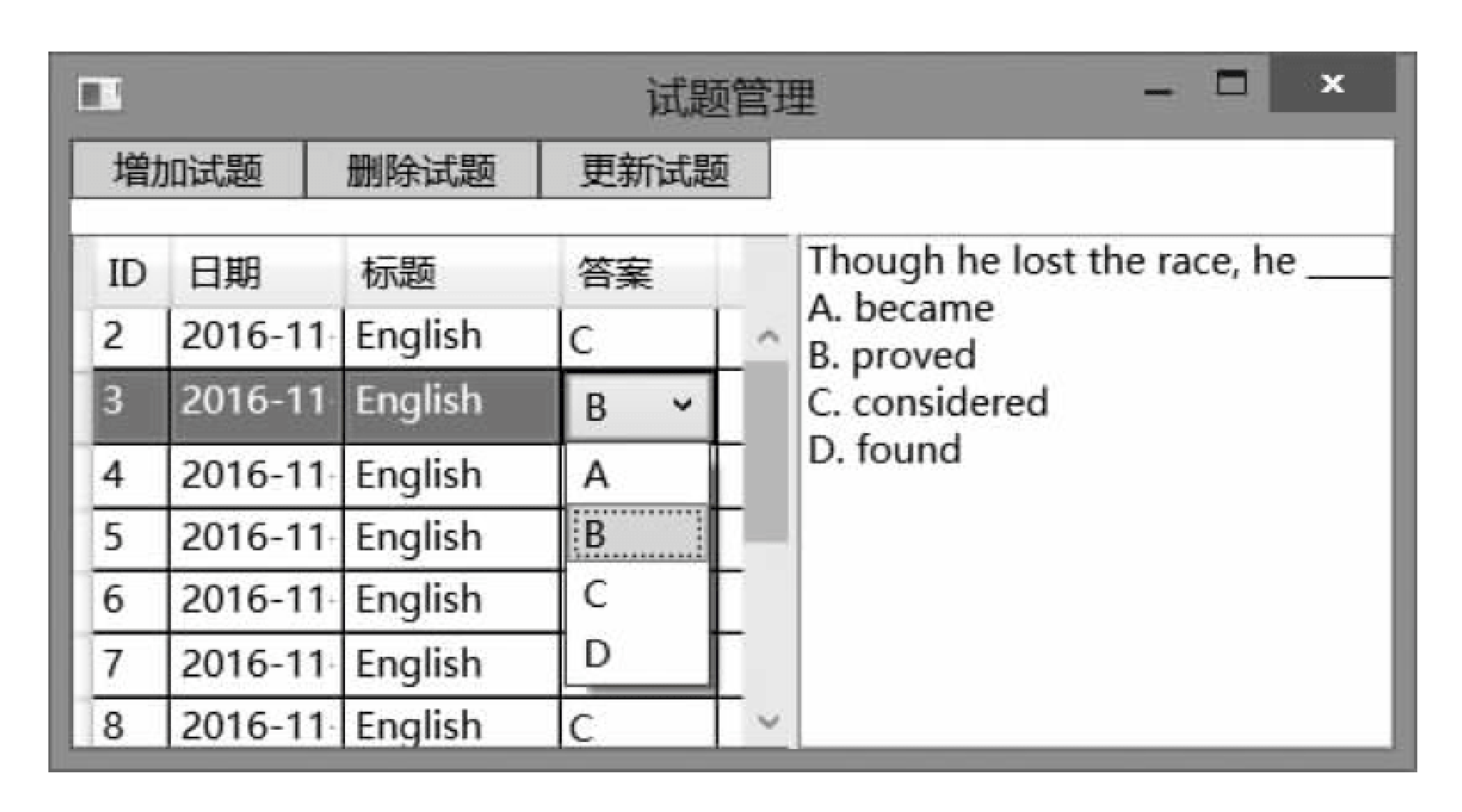


图 4-4 试题管理

普通用户登录后不可以实现试题管理。只可以进行测试练习与个人成绩管理操作。单击"测试练习"按钮后打开"测试练习"窗体,这个窗体中列出了随机的 10 个题目,每个题目下面有一个下拉列表框,用户从下拉列表框中选择 A、B、C、D 四个答案之一进行答题,完成后单击"提交答案"按钮,软件自动计算出成绩并把成绩上传到服务器,随后显示出标准答案供用户对比,如图 4-5 所示。

用户每做一次练习就生成一条成绩记录,普通用户可以查看与管理自己的测试成绩,不可以查看与管理别人的成绩,如图 4-6 所示。



图 4-5 测试练习



图 4-6 成绩管理

这个系统中客户端与服务器是通过 HTTP 协议(实际上是 SOAP 协议)进行通信的,因此服务器可以部署在远程的网络上,可以绕过一般防火墙,这种架构的软件系统实用性很强,本项目就以这个程序为例讲解这种程序的结构与设计方法。

4.1 用户注册登录

4.1.1 案例展示

设计控制台的服务器程序,用来管理用户的登录或者注册。设计 WPF 客户端窗体

程序连接服务器,输入用户名称与密码实现用户注册或者登录。如果用户不存在,则用这个用户名执行注册;如果用户已经存在,则执行登录。用户注册登录窗体如图 4-7 所示。



图 4-7 用户注册登录

4.1.2 技术要点

1. 数据库表

在 SQL Server 中建立数据库 NetTest,然后在数据库中建立 users 表。users 表是用户信息表,它有两个字段,一个是 uName 为用户名称,一个是 uPass 为用户密码,执行 SQL 命令建立 users 表:

```
create table users
(
    uName varchar(32) primary key,
    uPass varchar(512)
)
```

其中 uName 是表格的关键字段,用户名称不能重复。

2. WCF 服务

Windows Communication Foundation(WCF)是由微软公司开发的一系列支持数据通信的应用程序框架,翻译为 Windows 通信开发平台。WCF 是对于 ASMX、.NET Remoting、Enterprise Service、WSE、MSMQ 等技术的整合,也就是说 WCF 是 Web Service 的增强版。WCF 最基本的通信机制还是 SOAP,这就保证了系统之间的互操作性,如图 4-8 所示。一般来说 WCF 与 Web Service 相比有下面一些特性:

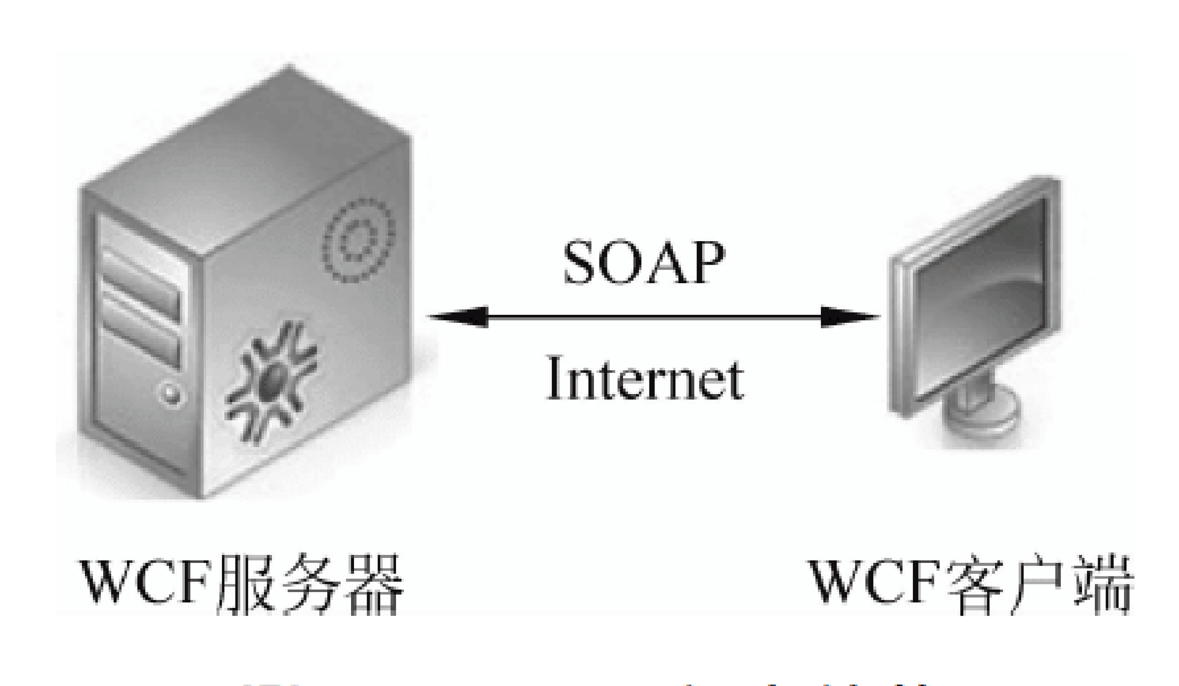


图 4-8 WCF 程序结构

(1) WCF 服务器可以是一个独立的 Windows 程序,或者 Windows Service,可以不依赖于 IIS。

(2) WCF 不但支持 HTTP 协议,还支持 HTTP、TCP/UDP、MSMQ、命名管道等协议。

(3) WCF 可以配置 BasicHttpBinding 来兼容(或者说变身成)前面讲过的 Web Service。

(4) WCF 的可配置性比 Web Service 强,很多功能特性可以通过 App.config 文件来配置。

(5) WCF 可以与 ASP.NET 集成、共享一个上下文(HttpContext)。

(6) WCF 的安全性比 Web Service 强,安全性的选择方案也多。

(7) WCF 支持多种会话模式:单向、双向、请求/响应。

(8) WCF 支持多种格式化方式,如 DataContractSerializer、XmlSerializer、DataContractJsonSerializer 等。

总的来说,WCF 在各个方面都比 Web Service 强,但是由于 WCF 是多种技术的集成,比 Web Service 复杂很多,涉及的知识很多,限于篇幅,本教程只对 WCF 的基本应用

进行讲解。

4.1.3 服务器程序

1. 3层架构

按以下的操作步骤建立一个3层架构的服务器程序：

(1) 建立一个空的解决方案项目 NetTestServer。

(2) 建立名称为 NetTestDAL 的 DAL 层类库，在其中建立数据库基本操作类文件 DBHelper.cs 以及用户注册管理类文件 UserService.cs。

(3) 建立名称为 NetTestBLL 的 BLL 层类库，其中建立用户管理类文件 UserManager.cs。

(4) 建立名称为 NetTestModel 的 Model 类库，其中建立数据类文件 ModelClass.cs。

(5) 建立名称为 NetTestServer 的控制台程序。

(6) 添加 NetTestServer 控制台程序对 NetTestBLL、NetTestModel 的引用，添加 NetTestBLL 对 NetTestDAL、NetTestModel 的引用，添加 NetTestDAL 对 NetTestModel 的引用。

(7) 在 ModelClass.cs 中设计一个 UserClass 用户类，它与数据库表 users 对应：

```
public class UserClass
{
    public String uName { get; set; }
    public String uPass { get; set; }
}
```

(8) 在 UserService.cs 设计 login 函数，这个函数以 UserClass 为参数，完成用户的注册或者登录，如果用户没有注册过就执行注册操作，如果已经注册过就执行登录操作。

```
public class UserService
{
    public String login(UserClass user)
    {
        String s="failed";
        using(SqlConnection con=new SqlConnection(DBHelper.conString))
        {
            DBHelper DB=new DBHelper(con);
            SqlParameter pName=new SqlParameter("@uName", SqlDbType.Char);
            pName.Value=user.uName;
            SqlParameter pPass=new SqlParameter("@uPass", SqlDbType.Char);
            pPass.Value=user.uPass;
            try
            {
                DB.executeCommand("insert into users values (@uName,@uPass)",
                pName, pPass);
                s="registered";
            }
            catch
```

```
            {
                if(DB.getScalar("select count ( * ) from users where uName= @
               uName and uPass=@uPass", pName, pPass)>0) s="logined";
            }
            con.Close();
        }
        return s;
    }
}
```

(9) 在 UserManager 设计对应的 login 函数，以便在 NetTestDAL 与它的上一层之间起到桥梁作用。

```
public class UserManager
{
    public String login(UserClass user)
    {
        UserService service=new UserService();
        return service.login(user);
    }
}
```

2. 建立 WCF 服务

1) 增加 WCF 服务器

接下来建立一个 WCF 服务，右击 NetTestServer 控制台项目，弹出快捷菜单，选择“添加新项”命令，弹出“添加新项”对话框，在对话框中选择“WCF 服务”，输入名称例如 NetTestService. cs 后确定，如图 4-9 所示。接下来在解决方案资源管理器中的 NetTestServer 下看到一个 NetTestService. cs 的文件和一个 INetTestService. cs 的文

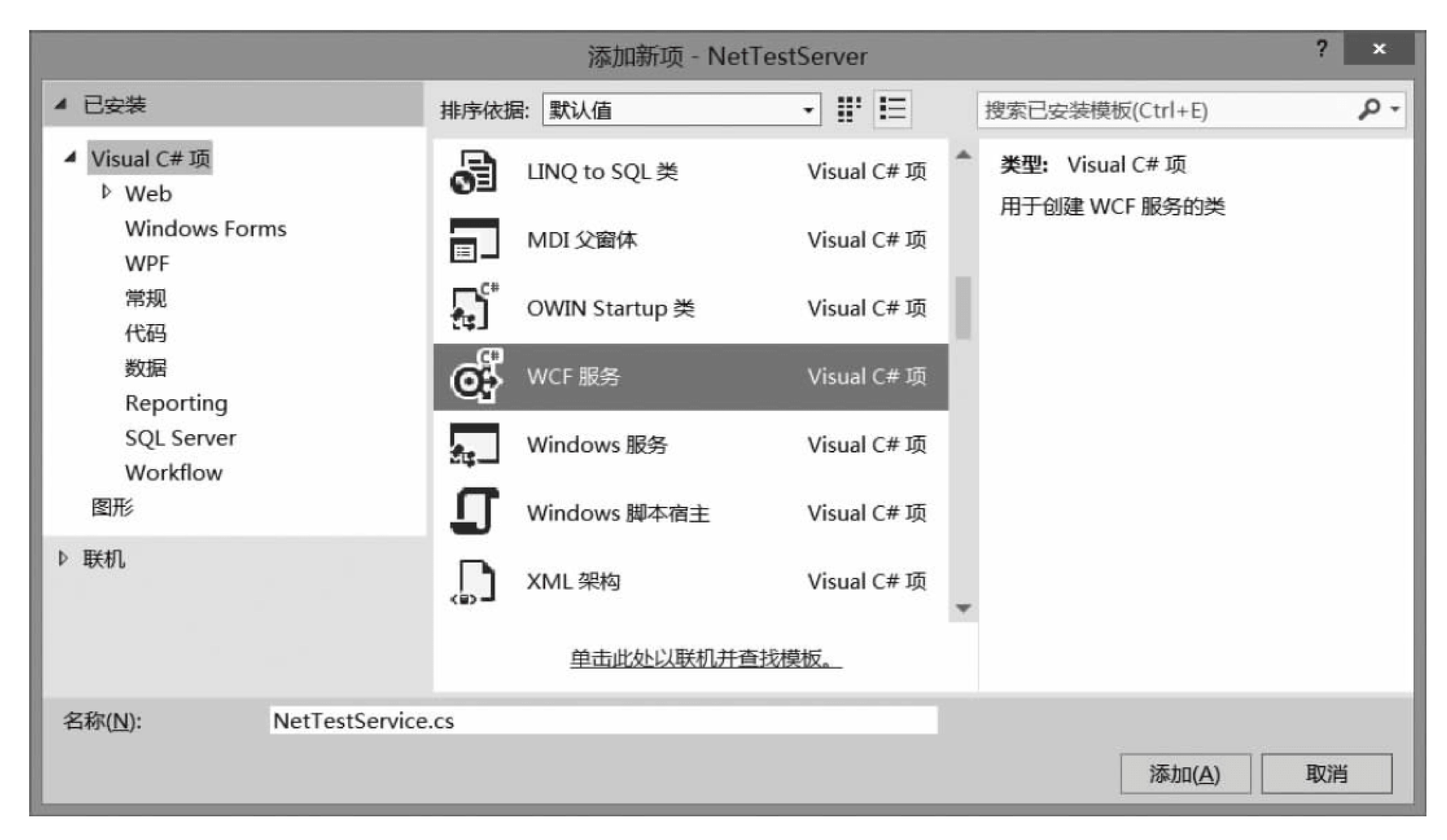


图 4-9 增加 WCG 服务

件，到此为止解决方案资源管理器如图 4-10 所示。

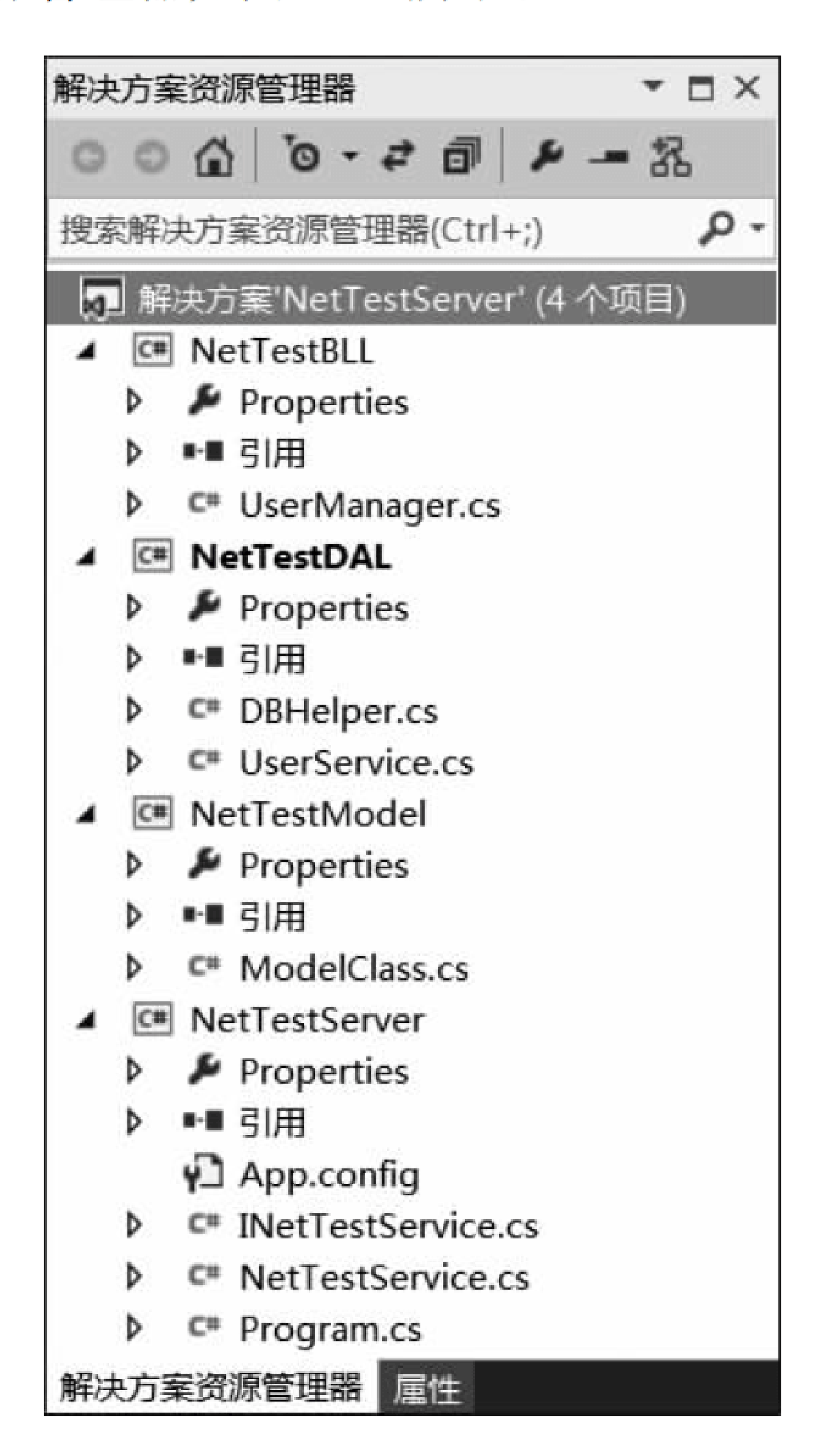


图 4-10　解决方案结构

打开 INetTestService. cs，可以看到其中定义了一个 INetTestService 的接口：

```
[ServiceContract]
public interface INetTestService
{
    [OperationContract]
    void DoWork();
}
```

这个接口有一个[ServiceContract]的属性，该属性表示该接口是用于公开服务的接口，类似 Web Service 中的[WebMethod]属性。接口中一个带有[OperationContract]属性的函数 DoWork，表示该函数是公开的服务函数。DoWork 函数是没有实现的，打开对应的文件 NetTestSevice. cs 可以看到有一个名称为 NetTestService 的类实现这个接口：

```
public class NetTestService : INetTestService
{
    public void DoWork()
    {
    }
}
```

DoWork 函数是对 INetTestService 接口函数的实现，但目前这个函数没有什么意义。我们把 DoWork 函数换成自己的 login 函数，接下来改变 INetTestService 接口如下：

```
[ServiceContract]
public interface INetTestService
{
    [OperationContract]
    String login(UserClass user);
}
```

对应地在 NetTestService.cs 的类中实现该函数：

```
public class NetTestService : INetTestService
{
    public String login(UserClass user)
    {
        UserManager manager=new UserManager();
        return manager.login(user);
    }
}
```

2）配置 App.config 文件

在 NetTestServer 中有一个 App.config 的文件，这是 WCF 的一个关键文件，这个文件是自动产生的一个 XML 格式文件，主要结构如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.serviceModel>
        <behaviors>
            <serviceBehaviors>
                <behavior name="NetTestBehavior">
                    <serviceMetadata httpGetEnabled="true" httpsGetEnabled=
                    "true" />
                    <serviceDebug includeExceptionDetailInFaults="false" />
                </behavior>
            </serviceBehaviors>
        </behaviors>
        <services>
            <service name="NetTestServer.NetTestService" behaviorConfigur-
            ation="NetTestBehavior">
                <endpoint address="" binding="basicHttpBinding" contract=
                "NetTestServer.INetTestService">
                    <identity>
                        <dns value="localhost" />
                    </identity>
```

```
            </endpoint>
            <endpoint address="mex" binding="mexHttpBinding" contract=
           "IMetadataExchange" />
        </service>
    </services>
</system.serviceModel>
</configuration>
```

这个文件的内容一般比较多，但是有几个重要的节点是必须关注的，而且有些地方还要做适当的修改。

<behavior>是行为节点，可以为它设置一个名字，例如 NetTestBehavior。其下面的<serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />表示允许元数据，也就是说在客户端可以发现这个服务，并得到服务信息。

<serviceDebug includeExceptionDetailInFaults="true" />表示服务器的错误可以传递到客户端，这在调试程序时很重要，可以随时让开发人员找到程序的错误。而在发布程序时可以把 true 改为 false，防止用户偷窥服务器的信息。

<service name="NetTestServer. NetTestService" behaviorConfiguration="NetTestBehavior">是一个服务节点，名称必须是程序中添加的 WCF 服务的名称 NetTestService，behaviorConfiguration 必须是<behavior>节点中定义的名称 NetTestBehavior。

<endpoint address="" binding="basicHttpBinding" contract="NetTestServer. INetTestService">表示 WCF 服务采用基本的 HTTP 协议，所用的协议接口是 INetTestService。

<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />是元数据交换协议，这个在开发阶段十分重要，否则客户端找不到服务器的元数据，不能正确生成客户端代理程序。

3）编写服务器程序

接下来要设计控制台程序，并让它包含这个 WCF 服务。在 Program. cs 中设计程序如下：

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            String url="http://localhost:8888/NetTestService/";
            ServiceHost host=new ServiceHost(typeof(NetTestService), new Uri
            (url));
            host.Open();
            Console.WriteLine(url+" 正在监听……");
        }
```

```
            catch(Exception exp)
            {
                Console.WriteLine(exp.Message);
            }
            Console.ReadKey();
        }
    }
```

url= "http://localhost:8888/NetTestService/"是服务器的地址,8888 端口号是自己定义的,只要系统中别的程序没有占用就可以使用,这个端口号建议不要使用 80、8080 等系统常用的端口,否则会与系统程序冲突。NetTestService 是本项目自定义的目录名称,一般把它定义为与 WCF 类一样的名称,也可以是别的名称。

host 是一个 ServiceHost 对象,这个对象就是服务器启动用的重要对象,建立它时调用 ServiceHost 的构造函数,函数的第一个参数是一个 type 类型, NetTestService 是 WCF 服务的类名称,不能是别的名称。第二个参数是服务器的 Uri 地址对象。host 建立后执行 Open 方法就使得服务器开始工作了,这个控制台程序的作用就是启动 WCF 服务。

3. 调试 WCF 服务

程序启动执行后就开始监听 http 协议的 8888 端口,结果如图 4-11 所示。客户端通过访问 http://localhost:8888/NetTestService/这个地址就可以与这个服务器通信,但是与 Web Service 一样,这个网址不是普通的网页,它是一个服务。打开浏览器在地址栏中输入服务器的监听地址,就可以看到图 4-12 所示的结果,只要看见"已创建服务"就表示服务器的服务已经创建成功。



图 4-11 启动服务器



图 4-12 浏览服务

4.1.4 客户端程序

1. 发现服务

首先要运行服务器程序 NetTestServer 使得服务器处于监听状态，然后使用 Visual Studio 新建一个 WPF 的窗体项目，命名为 NetTestClient。执行"添加服务引用"菜单命令(注意不是"添加引用"命令)，弹出"添加服务引用"对话框。在"地址"中输入服务器地址 http://localhost:8888/NetTestService/，然后单击"转到"按钮，结果可以看到"服务"中出现了 NetTestSrvice 的服务，再次打开这个服务器，在"操作"中看到服务器公开的 login 函数。在"命名空间"中输入一个名称，例如 WCF，如图 4-13 所示。然后单击"高级"按钮弹出"服务引用设置"对话框，选择"生成异步操作"单选按钮，如图 4-14 所示。单击"确定"按钮关闭服务引用设置对话框，回到"添加服务引用"对话框后再单击"确定"按钮关闭对话框。

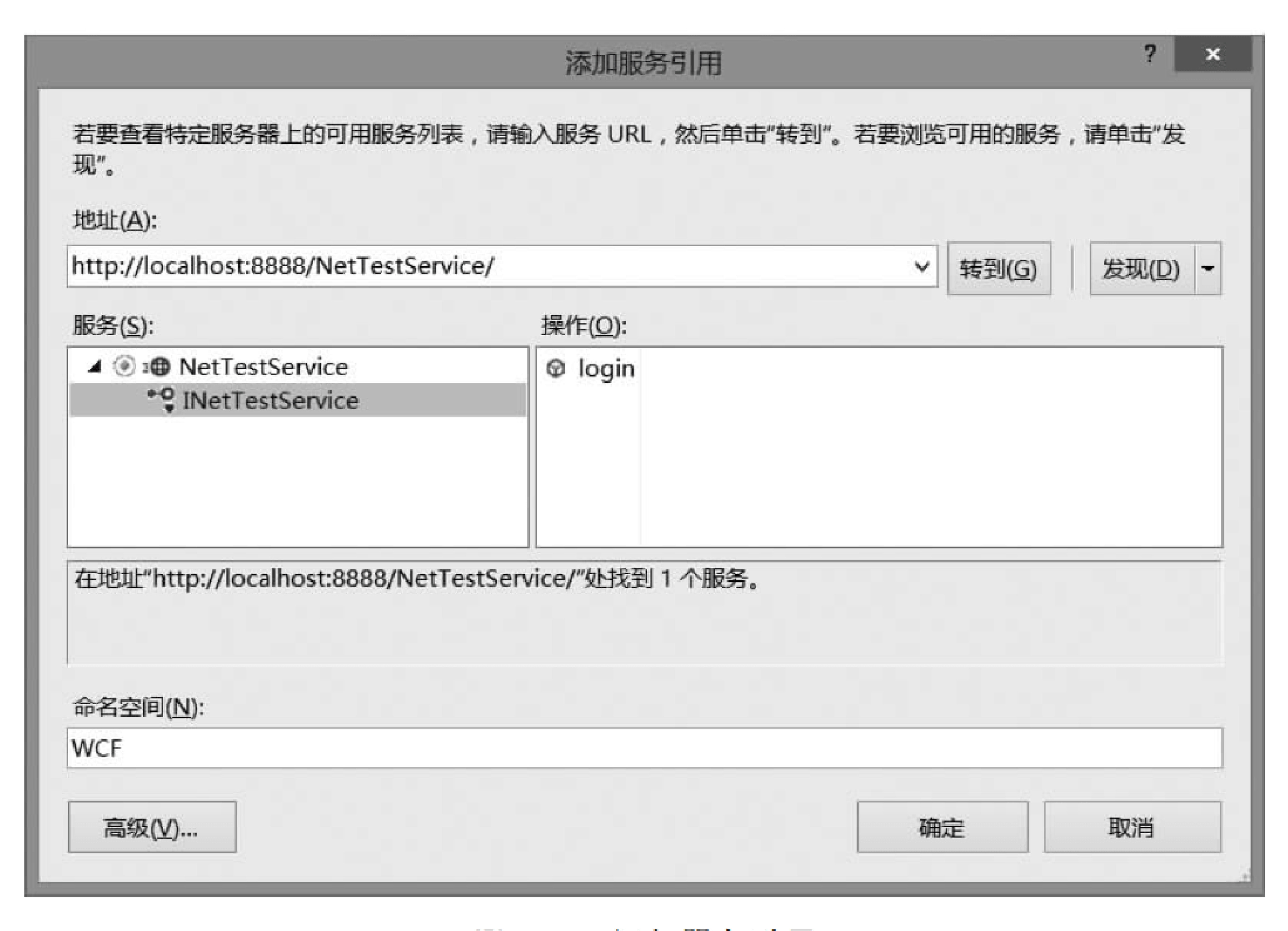


图 4-13 添加服务引用

客户端发现服务器的服务后会自动生成一个代理，在解决方案资源管理器中可以看到有一个名称为 Service References 的服务，该服务下的服务名称就是前面输入的 WCF，如图 4-15 所示。

2. 调用服务

1) 界面设计

客户端的界面很简单，在 WPF 程序的窗体上放一个名为 txtUser 的文本框用于输入用户名，一个名为 txtPass 的密码框用于输入密码，再放一个名为 btLogin 的按钮

服务引用设置

客户端

生成的类的访问级别(C): Public

☑ 允许生成异步操作(P)

○ 生成基于任务的操作(T)

◉ 生成异步操作(G)

数据类型

☐ 始终生成消息约定(M)

集合类型(L): System.Array

字典集合类型(D): System.Collections.Generic.Dictionary

☑ 重新使用引用的程序集中的类型(R)

◉ 重新使用所有引用的程序集中的类型(A)

○ 重新使用所引用的指定程序集中的类型(S):

☐ Microsoft.CSharp
☐ mscorlib
☐ PresentationCore
☐ PresentationFramework
☐ System
☐ System.Core
☐ System.Data
☐ System.Data.DataSetExtensions
☐ System.Xaml

兼容性

添加 Web 引用而不是服务引用。这将基于 .NET Framework 2.0 Web 服务技术生成代码。

添加 Web 引用(W)...

确定 取消

图 4-14 生成异步操作

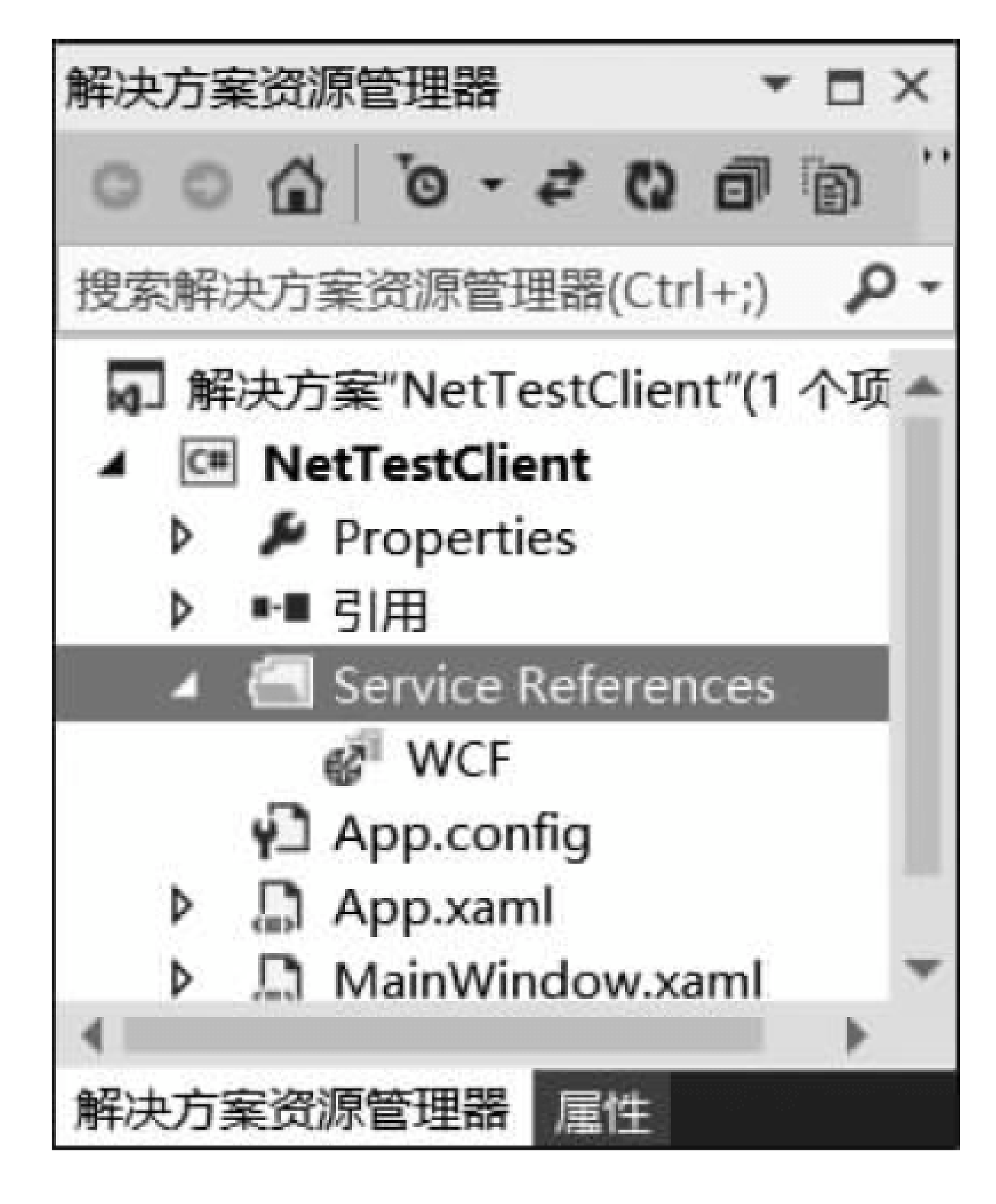


图 4-15 WCF 服务引用

Button 实现注册与登录，主要 XAML 代码如下：

```
<TextBox x:Name="txtUser" Text="xxx" HorizontalAlignment="Left" Height=
```

```
"23" Margin="42,10,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width=
"74"/>
    <PasswordBox x:Name="txtPass" Password="123"  HorizontalAlignment=
"Left" VerticalAlignment="Top" Margin="42,38,0,0" Width="74"/>
    <Button x:Name="btLogin" Content="注册登录" HorizontalAlignment="Left"
VerticalAlignment="Top" Width="75" Margin="10, 61, 0, 0" Click="btLogin_
Click"/>
    <TextBlock HorizontalAlignment="Left" x:Name="txtMsg" TextWrapping=
"Wrap" Text="TextBlock" VerticalAlignment="Top" RenderTransformOrigin=
"0.978,2.118" Margin="90,61,0,0"/>
```

2）程序设计

客户端程序的核心就是建立一个客户端去异步调用 login 函数实现用户的注册或者登录操作，程序代码如下：

```
namespace NetTestClient
{
    public partial class MainWindow : Window
    {
        WCF.NetTestServiceClient client;
        String url="http://localhost:8888/NetTestService/";
        public MainWindow()
        {
            InitializeComponent();
            //建立 client 对象
            client=new WCF.NetTestServiceClient();
            //设置异步函数
            client.loginCompleted+=client_loginCompleted;
            //设置访问的服务器地址
            client.Endpoint.Address=new System.ServiceModel.EndpointAddress
            (new Uri(url,UriKind.Absolute));
        }
        void client_loginCompleted(object sender, WCF.loginCompletedEventArgs e)
        {
            if(e.Error==null) txtMsg.Text=e.Result;
            else showMsg(e.Error.Message);
        }
        void showMsg(String s)
        {
            MessageBox.Show(s, "Information", MessageBoxButton.OK);
        }
        String encryptString(String s)
        {
            MD5 md5=new MD5CryptoServiceProvider();
            byte[] buf=Encoding.UTF8.GetBytes(s);
```

```
                buf=md5.ComputeHash(buf);
                s="";
                foreach (byte x in buf) s=s+x.ToString("X2");
                return s;
            }
            private void btLogin_Click(object sender, RoutedEventArgs e)
            {
                String uName=txtUser.Text.Trim();
                String uPass=txtPass.Password.Trim();
                try
                {
                    uPass=encryptString(uPass);
                    //异步调用 login 函数
                    client.loginAsync(new WCF.UserClass{uName=uName,uPass=
                    uPass});
                }
                catch(Exception exp) { showMsg(exp.Message); }
            }
        }
    }
```

程序首先通过 WCF 命名空间的 NetTestServiceClient 建立一个 client 对象，然后调用设置 login 的异步调用返回函数 client_loginCompleted，设置 client 访问的服务器地址 Endpoint.Address，在 btLogin_Click 中直接调用 loginAsync 异步函数。在调用完 loginAsync 后，客户端就把用户名称 uName 与密码 uPass 通过 SOAP 协议发送给服务器，服务器接收后就到数据库中去比对这个用户的信息，实行注册或者登录操作，然后返回执行结果。服务器的结果返回后，客户端就调用 client_loginCompleted 函数，该函数的 e.Result 就是服务器返回的结果。

运行服务器程序 NetTestServer 使其处于监听状态，运行客户端程序 NetTestClient 就可以实现用户的注册或者登录。

4.1.5　拓展训练

一个 WCF 服务器一旦公开了自己的函数接口，客户端就可以发现它并生成一个代理来调用服务器的接口函数。这个工作一般在开发客户端程序时执行，一旦开发完毕，为了防止别的开发人员恶意调用服务器的接口函数，需要关闭服务器的接口函数信息，使得用户不能再发现这个服务并建立代理。

可以通过设置服务器的 App.config 来保护 WCF 的服务。在 App.config 中，设置<serviceMetadata>节点为

```
<serviceMetadata httpGetEnabled="false"  httpsGetEnabled="false" />
```

同时在 App.config 中删除节点：

```
<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
```

经过这样修改后重新建立服务器程序 NetTestServer，再执行客户端程序 NetTestClent，结果发现不会影响客户端程序的正常执行。

但是再用浏览器浏览这个服务器地址，结果如图 4-16 所示，我们看到服务器禁止了元数据发布。再次用 Visual Studio 执行“添加服务引用”命令后，已经查找不到服务器的服务，如图 4-17 所示。



图 4-16　禁止元数据发布

添加服务引用

若要查看特定服务器上的可用服务列表，请输入服务 URL，然后单击“转到”。若要浏览可用的服务，请单击“发现”。

地址(A):

http://localhost:8888/NetTestService/　转到(G)　发现(D)

服务(S):　操作(O):

尝试在”http://localhost:8888/NetTestService/”处查找服务时发生 (详细信息) 错误。

命名空间(N):

ServiceReference1

高级(V)...　确定　取消

图 4-17　查找不到服务

由此可见，通过这种方法重新建立服务器程序进行发布，客户端开发人员已经不能再获取服务的信息了，也就是说不能生成代理去调用服务器函数了。这个方法有效地保护了 WCF 的服务免遭恶意的调用，同时又不会影响原来客户端程序的运行。如果需要再次开发客户端，就把 App.config 再修改成原来的那样，重新编译 NetTestServer 即可。

4.2　试题记录增加

4.2.1　案例展示

设计服务器的程序管理测试试题，客户端窗体程序能够以管理者 Admin 的身份完成试题的上传。一个试题包括试题的标题、内容与答案，标题用一个单行文本框 TextBox 输入，内容用多行的文本框 TextBox 输入，答案用下拉列表框 ComboBox 设置，如图 4-18 所示。



图 4-18　增加试题

4.2.2　技术要点

1. 数据库表

试题数据库表 tests 包含一个自动增长列 ID、试题的标题 tTitle、试题答案 tAnswer、试题上传日期 tDate 与试题内容 tText，其中 tText 是 text 类型，用于存储任意长的文本数据。tests 表的 SQL 命令如下：

```
create table tests
(
    ID int identity(1,1) primary key,
    tTitle varchar(256),
    tAnswer varchar(2),
    tDate varchar(32),
    tText text,
)
```

同时在 NetTestModel 中建立一个 TestClass 类与这个表对应：

```
public class TestClass
{
    public int ID { get; set; }
```

```
        public String tDate { get; set; }
        public String tTitle { get; set; }
        public String tAnswer { get; set; }
        public String tText { get; set; }
    }
```

2. 试题管理

在服务器程序的 NetTestDAL 增加一个 TestService 的类，这个类实现试题的上传管理，其中 addTest 函数是上传函数，一个试题上传成功后，就在 tests 表中增加一条记录，addTest 返回新增记录的 ID 号。

由于只有管理员才能上传试题，因此在 addTest 中要求提供用户名称 uName 与密码 uPass，调用身份验证函数 verifyAdmin 验证用户是否为管理员 Admin，如果是管理员，verifyAdmin 就返回 true，否则返回 false。

```
    namespace NetTestDAL
    {
        public class TestService
        {
            bool verifyAdmin(DBHelper DB, UserClass user)
            {
                if(user.uName=="Admin")
                {
                    SqlParameter pName=new SqlParameter("@uName", SqlDbType.
                    Char); pName.Value=user.uName;
                    SqlParameter pPass=new SqlParameter("@uPass", SqlDbType.
                    Char); pPass.Value=user.uPass;
                    return((int)DB.getScalar("select count(*) from users where
                    uName=@uName and uPass=@uPass", pName, pPass)>0);
                }
                return false;
            }
            public int addTest(UserClass user,ref TestClass test)
            {
                test.ID=0;
                using(SqlConnection con=new SqlConnection(DBHelper.conString))
                {
                    DBHelper DB=new DBHelper(con);
                    if(verifyAdmin(DB, user))
                    {
                        SqlParameter pDate=new SqlParameter("@tDate", SqlDbType.
                        Char); pDate.Value=DateTime.Now.ToString("yyyy-MM-dd HH:
                        mm:ss");
                        SqlParameter pTitle=new SqlParameter("@tTitle", SqlDbType.
```

```
                    Char); pTitle.Value=test.tTitle;
                    SqlParameter pText=new SqlParameter("@tText", SqlDbType.
                    Text); pText.Value=test.tText;
                    SqlParameter pAnswer=new SqlParameter("@tAnswer", SqlDbType.
                    Char); pAnswer.Value=test.tAnswer;
                    if(DB.executeCommand("insert into tests (tDate,tTitle,tText,
                    tAnswer) values (@ tDate, @ tTitle, @ tText, @ tAnswer)",
                    pDate, pTitle, pText,pAnswer)>0)
                    {
                        //插入成功时获取插入记录的 ID
                        test.ID=DB.getScalar("select top 1 ID from tests order
                        by ID desc");
                    }
                }
                con.Close();
            }
            return test.ID;
        }
    }
}
```

在 NetTestBLL 中再增加一个 TestManager 类，这个类直接调用 TestService 的 addTest 函数：

```
namespace NetTestBLL
{
    public class TestManager
    {
        public int addTest(UserClass user, ref TestClass test)
        {
            TestService service=new TestService();
            return service.addTest(user, ref test);
        }
    }
}
```

3. 用户验证

在客户端调用服务器的 WCF 服务时，需要把用户的名称 uName 与密码 uPass 传递给服务器实现用户验证。一种行之有效的方法是采用 SOAP 的 MessageHeader，用这个类的静态函数 CreateHeader 建立一个 MessageHeader 对象，例如：

```
MessageHeader user=MessageHeader.CreateHeader("uName", "MySpace", uName);
```

MessageHeader 是一种键/值(key/value)对的结构，其中第一个参数 uName 是

MessageHeader 的键(key),第二个对象是命名空间,命名空间的名称可以自己设定,第三个参考是键对应的值(value)。

MessageHeader 对象要加在一个 OperationContextScope 的局部范围内,典型的代码如下:

```
using (OperationContextScope scope = new OperationContextScope (client.
InnerChannel))
{
    MessageHeader user = MessageHeader.CreateHeader ( "uName", "MySpace",
    uName);
    MessageHeader pass = MessageHeader.CreateHeader ( "uPass", "MySpace",
    uPass);
    OperationContext.Current.OutgoingMessageHeaders.Add(user);
    OperationContext.Current.OutgoingMessageHeaders.Add(pass);
    //上传试题数据
    client.addTestAsync(test);
}
```

如果这样调用 addTestAsync 函数,用户名称 uName 与密码 uPass 就可以传递到服务器。在服务器中可以通过 GetHeader 函数取出指定命名空间中指定键的值,从而取出 uName 与 uPass 的值,方法如下:

```
String uName=OperationContext.Current.IncomingMessageHeaders.GetHeader
<String>("uName", "MySpace");
String uPass=OperationContext.Current.IncomingMessageHeaders.GetHeader
<String>("uPass", "MySpace");
```

有了这个方法后,客户端调用服务器的函数就可以把用户名称与密码同时传递给服务器,服务器要对用户进行验证,验证不通过时不允许调用服务器的函数,这是对服务器函数的保护方法之一。

4.2.3 服务器程序

1. INetTestService 接口

这个接口中定义了登录函数 login 与增加试题函数 addText,接口如下:

```
[ServiceContract]
public interface INetTestService
{
    [OperationContract]
    String login(UserClass user);
    [OperationContract]
    TestClass addTest(TestClass test);
}
```

2. NetTestService 类

增加试题函数 addTest 获取用户名称 uName 与密码 uPass，对用户进行验证，然后实现题目的存储，函数如下：

```
public class NetTestService : INetTestService
{
    UserClass getUser()
    {
        String uName=OperationContext.Current.IncomingMessageHeaders.
        GetHeader<String>("uName", "MySpace");
        String uPass=OperationContext.Current.IncomingMessageHeaders.
        GetHeader<String>("uPass", "MySpace");
        return new UserClass { uName=uName, uPass=uPass };
    }
    public TestClass addTest(TestClass test)
    {
        TestManager manager=new TestManager();
        manager.addTest(getUser(), ref test);
        return test;
    }
}
```

其中 getUser 函数获取用户的信息，addTest 函数完成试题的存储。

4.2.4 客户端程序

首先要运行服务器程序 NetTestServer 使得服务器处于监听状态，然后在客户端程序的解决方案资源管理器中找到 Service Reference 中的 WCF，右击 WCF，弹出快捷菜单，执行“更新服务引用”命令，客户端就能找到并更新服务引用。

1. 界面设计

客户端的界面很简单，在 WPF 程序的窗体上放一个名为 txtUser 的文本框用于输入用户名，放一个名为 txtPass 的密码框用于输入密码，再放一个名为 btUpload 的按钮 Button 实现试题上传，主要 XAML 代码如下：

```
<TextBox x:Name="txtUser" Text="Admin" HorizontalAlignment="Left" Height=
"23" Margin="42,10,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width=
"74"/>
<PasswordBox x:Name="txtPass" Password="123"  HorizontalAlignment="Left"
VerticalAlignment="Top" Margin="159,12,0,0" Width="74"/>
<Button x:Name="btUpload" Content="试题上传" HorizontalAlignment="Left"
VerticalAlignment="Top" Width="75" Margin="253,111,0,0" Click="btUpload_
Click"/>
```

```
<TextBlock HorizontalAlignment="Left" x:Name="txtMsg" TextWrapping="Wrap"
Text="TextBlock" VerticalAlignment="Top" RenderTransformOrigin="0.978,
2.118" Margin="253,152,0,0"/>
<TextBlock HorizontalAlignment="Left" x:Name="txtMsg_Copy" TextWrapping=
"Wrap" Text="用户" VerticalAlignment="Top" RenderTransformOrigin="0.978,
2.118" Margin="10,11,0,0"/>
<TextBlock HorizontalAlignment="Left" x:Name="txtMsg_Copy1" TextWrapping=
"Wrap" Text="密码" VerticalAlignment="Top" RenderTransformOrigin="0.978,
2.118" Margin="121,14,0,0"/>
<TextBox x:Name="txtTitle" Text="XML 序列化" HorizontalAlignment="Left"
Height="23" Margin="42,39,0,0" TextWrapping="Wrap" VerticalAlignment="Top"
Width="191"/>
<TextBlock HorizontalAlignment="Left" x:Name="txtMsg_Copy2" TextWrapping=
"Wrap" Text="标题" VerticalAlignment="Top" RenderTransformOrigin="0.978,
2.118" Margin="10,41,0,0"/>
<TextBox x:Name="txtText" HorizontalAlignment="Left" Height="116"
VerticalAlignment="Top" Width="217" Margin="16,67,0,0" AcceptsReturn=
"True" HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility=
"Auto">
</TextBox>
<ComboBox x:Name="cbAnswer" HorizontalAlignment="Left" VerticalAlignment=
"Top" Width="55" Margin="282,67,0,0"/>
<TextBlock HorizontalAlignment="Left" x:Name="txtMsg_Copy3" TextWrapping=
"Wrap" Text="答案" VerticalAlignment="Top" RenderTransformOrigin="0.978,
2.118" Margin="253,67,0,0"/>
```

2. 程序设计

在客户端更新服务引用时就建立了服务器试题类 TestClass 的客户端代理类 WCF. TestClass 与代理函数 addTestAsync，因此直接生成一个 WCF. TestClass 对象，调用 addTestAsync 函数即可，程序如下：

```
public partial class MainWindow : Window
{
    WCF.NetTestServiceClient client;
    String url="http://localhost:8888/NetTestService/";
    public MainWindow()
    {
        InitializeComponent();
        //建立 client 对象
        client=new WCF.NetTestServiceClient();
        //设置异步函数
        client.addTestCompleted+=client_addTestCompleted;
        //设置访问的服务器地址
```

```
        client.Endpoint.Address=new System.ServiceModel.EndpointAddress
        (new Uri(url,UriKind.Absolute));
        //设置答案列表
        cbAnswer.Items.Add("A");
        cbAnswer.Items.Add("B");
        cbAnswer.Items.Add("C");
        cbAnswer.Items.Add("D");
        cbAnswer.SelectedIndex=0;
    }
    void client_addTestCompleted(object sender, WCF.addTestCompletedEventArgs e)
    {
        if(e.Error==null)
        {
            WCF.TestClass test=e.Result;
            if(test.ID>0) txtMsg.Text="上传成功 ID="+test.ID.ToString();
        }
        else showMsg(e.Error.Message);
    }
    void showMsg(String s)
    {
        MessageBox.Show(s, "Information", MessageBoxButton.OK);
    }
    String encryptString(String s)
    {
        MD5 md5=new MD5CryptoServiceProvider();
        byte[] buf=Encoding.UTF8.GetBytes(s);
        buf=md5.ComputeHash(buf);
        s="";
        foreach (byte x in buf) s=s+x.ToString("X2");
        return s;
    }
    private void btUpload_Click(object sender, RoutedEventArgs e)
    {
        String uName=txtUser.Text.Trim();
        String uPass=txtPass.Password.Trim();
        String tTitle=txtTitle.Text.Trim();
        String text=txtText.Text.Trim();
        if(uName !="" && uPass !="" && tTitle !="" && text !="")
        {
            try
            {
                uPass=encryptString(uPass);
                using(OperationContextScope scope=new OperationContextScope
                (client.InnerChannel))
                {
                    MessageHeader user=MessageHeader.CreateHeader("uName",
```

```
                        "MySpace", uName);
                        MessageHeader pass=MessageHeader.CreateHeader("uPass",
                        "MySpace", uPass);
                        OperationContext.Current.OutgoingMessageHeaders.Add(user);
                        OperationContext.Current.OutgoingMessageHeaders.Add(pass);
                        //上传试题数据
                        WCF.TestClass test = new WCF.TestClass { ID = 0, tAnswer =
                        cbAnswer.SelectedItem.ToString(), tTitle = tTitle, tText =
                        text };
                        client.addTestAsync(test);
                    }
                }
                catch(Exception exp) { showMsg(exp.Message); }
            }
        }
    }
```

4.2.5 拓展训练

在服务器 App. config 中一般设置<serviceDebug>节为

```
<serviceDebug includeExceptionDetailInFaults="true" />
```

这个设置意味着在服务器有异常出现时，这个异常可以被客户端捕捉到。

读者可以尝试更改数据库表 tests 的名称，那么服务器执行时必定找不到这个表，会在客户端抛出一个异常，如图 4-19 所示。



图 4-19　捕捉异常

如果服务器设置<serviceDebug>节为

```
<serviceDebug includeExceptionDetailInFaults=
"false" />
```

那么在客户端就捕捉不到服务器的异常。

一般在程序调试阶段把这个值设置为 true，以便发现服务器的异常，帮助开发人员发现问题，一旦开发完毕开始部署服务器，建议把它设置为 false，避免客户端用户偷窥到服务器的信息。

4.3 试题记录删除

4.3.1 案例展示

设计服务器管理试题，一个试题包括实体编号、标题、上传日期、答案、试题内容等属性，客户端连接后能对试题进行浏览、删除等操作，如图 4-20 所示。删除时选择一行，单

击"删除试题"按钮,这一选择试题就会被删除。

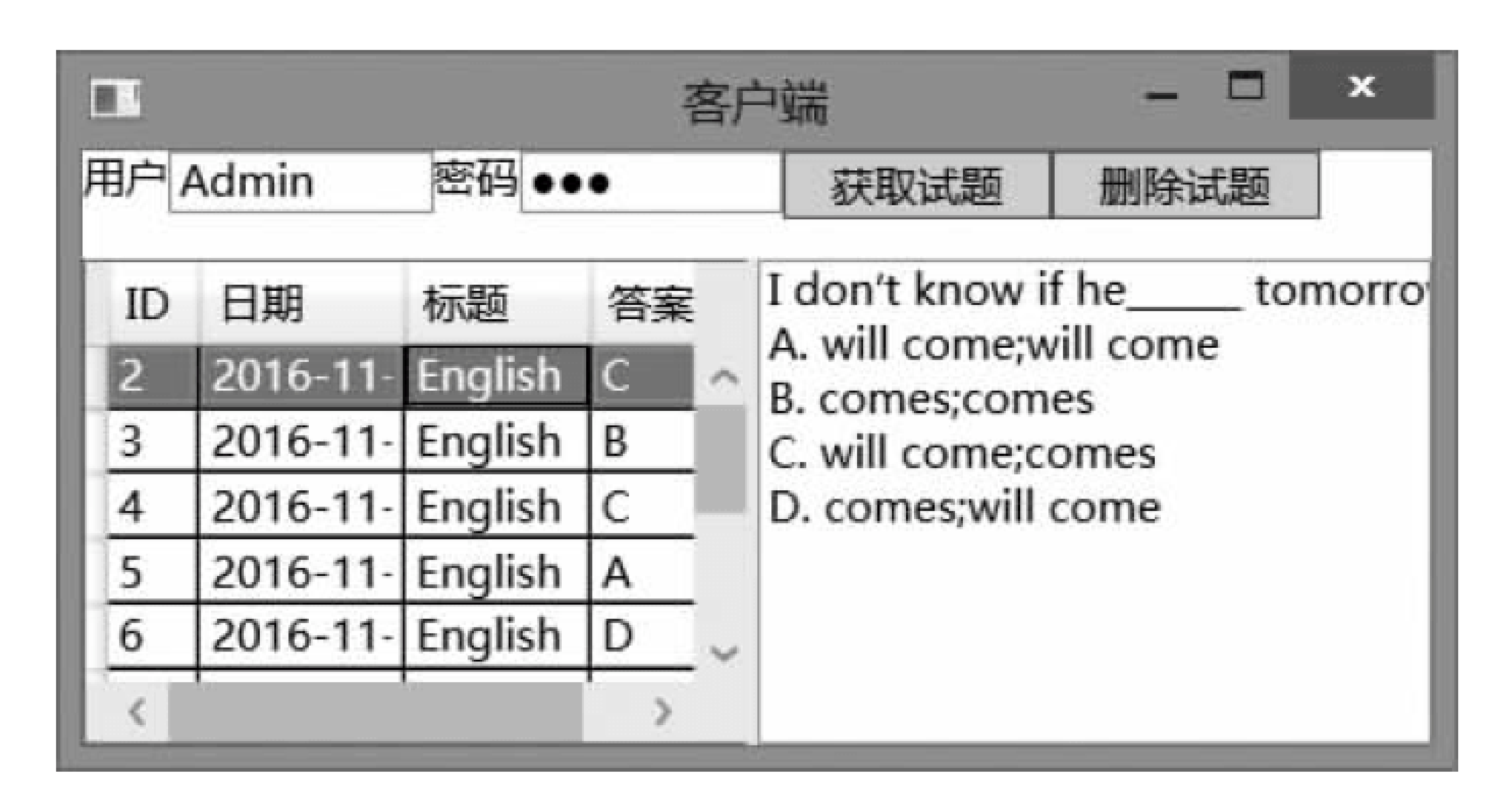


图 4-20 试题浏览

4.3.2 技术要点

在服务器程序的 TestService 类中增加试题删除函数 deleteTest,这个函数以试题类 TestClass 作为参数,另外增加 getTestDataTable 函数获取试题的数据集。所有这些操作都必须经过管理员用户身份验证,只有管理员可以进行试题的管理。

```
namespace NetTestDAL
{
    public class TestService
    {
        bool verifyAdmin(DBHelper DB, UserClass user)
        {
            if(user.uName=="Admin")
            {
                SqlParameter pName = new SqlParameter("@uName", SqlDbType.
                Char); pName.Value=user.uName;
                SqlParameter pPass = new SqlParameter("@uPass", SqlDbType.
                Char); pPass.Value=user.uPass;
                return((int)DB.getScalar("select count(*) from users where
                uName=@uName and uPass=@uPass", pName, pPass)>0);
            }
            return false;
        }
        public DataTable getTestDataTable(UserClass user)
        {
            DataTable dt=null;
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                if(verifyAdmin(DB, user))
                {
```

```
                SqlDataAdapter adapter = DB.getAdapter("select ID,tDate,
                tTitle,tAnswer,tText from tests order by ID");
                dt=new DataTable("Test");
                adapter.Fill(dt);
            }
            con.Close();
        }
        return dt;
    }
    public bool deleteTest(UserClass user,TestClass test)
    {
        bool flag=false;
        using(SqlConnection con=new SqlConnection(DBHelper.conString))
        {
            DBHelper DB=new DBHelper(con);
            if(verifyAdmin(DB, user))
            {
                if(DB.executeCommand("delete from tests where ID="+
                test.ID.ToString())>0) flag=true;
            }
            con.Close();
        }
        return flag;
    }
}
```

在 NetNetTestBLL 中编写对应的管理函数，它们与 TestService 中的函数相对应，完成试题数据集的获取、试题的删除等操作。

```
namespace NetTestBLL
{
    public class TestManager
    {
        public DataTable getTestDataTable(UserClass user)
        {
            TestService service=new TestService();
            return service.getTestDataTable(user);
        }
        public bool deleteTest(UserClass user,TestClass test)
        {
            TestService service=new TestService();
            return service.deleteTest(user, test);
        }
    }
}
```

4.3.3 服务器程序

1. INetTestService 接口

```
[ServiceContract]
public interface INetTestService
{
    [OperationContract]
    DataTable getTestDataTable();
    [OperationContract]
    bool deleteTest(TestClass test);
}
```

2. NetTestService 类

```
public class NetTestService : INetTestService
{
    UserClass getUser()
    {
        String uName=OperationContext.Current.IncomingMessageHeaders.
        GetHeader<String>("uName", "MySpace");
        String uPass=OperationContext.Current.IncomingMessageHeaders.
        GetHeader<String>("uPass", "MySpace");
        return new UserClass { uName=uName, uPass=uPass };
    }
    public DataTable getTestDataTable()
    {
        TestManager manager=new TestManager();
        return manager.getTestDataTable(getUser());
    }
    public bool deleteTest(TestClass test)
    {
        TestManager manager=new TestManager();
        return manager.deleteTest(getUser(),test);
    }
}
```

4.3.4 客户端程序

运行服务器程序 NetTestServer 使得服务器处于监听状态，然后在客户端程序的解决方案资源管理器中找到 Service References 中的 WCF，右击 WCF，弹出快捷菜单，执行“更新服务引用”命令，客户端就能找到并更新服务引用。

1. 界面设计

客户端的 WPF 程序的窗体上放一个名为 uGrid 的 DataGrid 数据集显示控件显示试题数据集，放一个名称为 txtText 的 TexBox 控件显示试题内容。

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="30"/>
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <StackPanel Orientation="Horizontal" Grid.Row="0">
        <TextBlock HorizontalAlignment="Left"  TextWrapping="Wrap" Text="用
        户" VerticalAlignment="Top" />
        <TextBox x:Name="txtUser" Text="Admin" HorizontalAlignment="Left"
        VerticalAlignment="Top" Width="74"/>
        <TextBlock HorizontalAlignment="Left" Text="密码" VerticalAlignment
        ="Top" />
        <PasswordBox x:Name="txtPass" Password="123"  HorizontalAlignment
        ="Left" VerticalAlignment="Top"  Width="74"/>
        <Button x:Name="btGetTest" Content="获取试题" HorizontalAlignment=
        "Left" VerticalAlignment =" Top "  Width =" 75 "  Click =" btGetTest _
        Click"/>
        <Button x:Name="btDeleteTest" Content="删除试题" HorizontalAlignment
        ="Left" VerticalAlignment="Top" Width="75" Click="btDeleteTest_
        Click"/>
    </StackPanel>
    <Grid Grid.Row="1">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="1*" />
            <ColumnDefinition Width="1*" />
        </Grid.ColumnDefinitions>
        <DataGrid x:Name="uGrid" Grid.Column="0" AutoGenerateColumns=
        "False" IsReadOnly="True" SelectionMode="Single" SelectionChanged=
        "uGrid_SelectionChanged" CanUserDeleteRows="False" CanUserAddRows=
        "False" CanUserReorderColumns="False" CanUserResizeRows="False">
            <DataGrid.Columns>
                <DataGridTextColumn Binding="{Binding ID}" Header="ID" Width
                ="50" />
                <DataGridTextColumn Binding="{Binding tDate}" Header="日期"
                Width="100"  />
                <DataGridTextColumn Binding="{Binding tTitle}" Header="标题"
                Width="100" />
                <DataGridTextColumn Binding="{Binding tAnswer}" Header="答
                案" Width="50" />
```

```
            </DataGrid.Columns>
        </DataGrid>
        <GridSplitter Width="3" />
        <TextBox x:Name="txtText" Grid.Column="1"  Text="{Binding tText,
        Mode=TwoWay}" VerticalScrollBarVisibility="Auto" />
    </Grid>
</Grid>
```

2. 程序设计

客户端的 uGrid 的选择行发生变化时,触发 SelectionChanged 函数,在这个函数中找到对应的数据集记录,从而在 txtText 中显示该题目的文本内容。程序采用异步操作,在删除试题时在异步函数中传递试题对象到对应的完成函数中,在完成函数中找回这个试题对象,以便服务器更新后在客户端的数据集中更新该试题。

```
public partial class MainWindow : Window
{
    WCF.NetTestServiceClient client;
    String url="http://localhost:8888/NetTestService/";
    DataTable dt;
    DataView dv;
    public MainWindow()
    {
        InitializeComponent();
        //建立 client 对象
        client=new WCF.NetTestServiceClient();
        //设置异步函数
        client.getTestDataTableCompleted +=client_getTestDataTableCompleted;
        client.deleteTestCompleted+=client_deleteTestCompleted;
        //设置访问的服务器地址
        client.Endpoint.Address=new System.ServiceModel.EndpointAddress
        (new Uri(url, UriKind.Absolute));
    }
    void client_deleteTestCompleted(object sender,
    WCF.deleteTestCompletedEventArgs e)
    {
        if(e.Error==null)
        {
            DataRow row=dt.AsEnumerable().FirstOrDefault(x=>(int)x["ID"]==
            (int)e.UserState);
            if(row !=null)
            {
                row.Delete();
                //从数据集中删除这一行,但是不影响别的行的状态
```

```
                row.AcceptChanges();
            }
        }
        else showMsg(e.Error.Message);
    }
    void client_getTestDataTableCompleted(object sender,
    WCF.getTestDataTableCompletedEventArgs e)
    {
        if(e.Error==null)
        {
            dt=e.Result;
            dv=dt.DefaultView;
            uGrid.ItemsSource=dv;
        }
        else showMsg(e.Error.Message);
    }
    void showMsg(String s)
    {
        MessageBox.Show(s, "Information", MessageBoxButton.OK);
    }
    String encryptString(String s)
    {
        MD5 md5=new MD5CryptoServiceProvider();
        byte[] buf=Encoding.UTF8.GetBytes(s);
        buf=md5.ComputeHash(buf);
        s="";
        foreach (byte x in buf) s=s+x.ToString("X2");
        return s;
    }
    private void btGetTest_Click(object sender, RoutedEventArgs e)
    {
        //获取试题列表
        String uName=txtUser.Text.Trim();
        String uPass=txtPass.Password.Trim();
        if(uName !="" && uPass !="")
        {
            try
            {
                uPass=encryptString(uPass);
                using(OperationContextScope scope=new OperationContextScope
                (client.InnerChannel))
                {
                    MessageHeader user=MessageHeader.CreateHeader("uName",
                    "MySpace", uName);
```

```
                    MessageHeader pass=MessageHeader.CreateHeader("uPass",
                    "MySpace", uPass);
                    OperationContext.Current.OutgoingMessageHeaders.Add(user);
                    OperationContext.Current.OutgoingMessageHeaders.Add(pass);
                    client.getTestDataTableAsync();
                }
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
    private void uGrid_SelectionChanged(object sender,
    SelectionChangedEventArgs e)
    {
        if(uGrid.SelectedIndex>=0)
        {
            txtText.DataContext=dv[uGrid.SelectedIndex];
        }
        else txtText.DataContext=null;
    }
    private void btDeleteTest_Click(object sender, RoutedEventArgs e)
    {
        if(uGrid.SelectedIndex>=0)
        {
            int index=uGrid.SelectedIndex;
            WCF.TestClass test=new WCF.TestClass { ID=(int)dv[index]["ID"] };
            String uName=txtUser.Text.Trim();
            String uPass=txtPass.Password.Trim();
            if(uName !="" && uPass !="")
            {
                try
                {
                    uPass=encryptString(uPass);
                    using(OperationContextScope scope=new
                    OperationContextScope(client.InnerChannel))
                    {
                        MessageHeader user=MessageHeader.CreateHeader
                        ("uName", "MySpace", uName);
                        MessageHeader pass=MessageHeader.CreateHeader
                        ("uPass", "MySpace", uPass);
                        OperationContext.Current.OutgoingMessageHeaders.Add
                        (user);
                        OperationContext.Current.OutgoingMessageHeaders.Add
                        (pass);
                        client.deleteTestAsync(test, test.ID);
```

```
                    }
                }
                catch(Exception exp) { showMsg(exp.Message); }
            }
        }
    }
}
```

4.3.5 拓展训练

删除试题实际上可以批量删除，既先选择要删除的任意多条记录，然后单击“删除试题”把选择的记录全部删除。

1. 服务器批量删除

在服务器一端的 TestService 中把 deleteTest 函数改造一下，使得它接受一个 List<int>参数，这个参数中列出了所有要删除的记录的 ID 号，在 deleteTest 中逐条删除记录即可，函数如下：

```
public bool deleteTest(UserClass user,List<int>IDS)
{
    bool flag=false;
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyAdmin(DB, user))
        {
            foreach(int ID in IDS)
             if(DB.executeCommand("delete from tests where ID="+ID.ToString
             ())>0) flag=true;
        }
        con.Close();
    }
    return flag;
}
```

2. 客户端批量删除

客户端的 DataGrid 设置为多选的，就是把 SelectionMode 改成“Extended”：

```
<DataGrid x:Name="uGrid" Grid.Column="0" AutoGenerateColumns="False"
IsReadOnly="True" SelectionMode="Extended" SelectionChanged="uGrid_
SelectionChanged" CanUserDeleteRows="False" CanUserAddRows="False"
CanUserReorderColumns="False" CanUserResizeRows="False">
 ⋮
```

```
</DataGrid>
```

再修改客户端的删除函数，它找出所有选择要删除的试题的 ID 号的序列，调用 deleteTestAsync 函数就完成一批试题的删除。

```
private void btDeleteTest_Click(object sender, RoutedEventArgs e)
{
    if(uGrid.SelectedItems.Count>0)
    {
        String uName=txtUser.Text.Trim();
        String uPass=txtPass.Password.Trim();
        if(uName !="" && uPass !="")
        {
            //获取要删除的试题的 ID 序列
            List<int>IDS=new List<int>();
            for(int i=0;i<uGrid.SelectedItems.Count;i++)
            {
              DataRowView row=(DataRowView)uGrid.SelectedItems[i];
              IDS.Add((int)row["ID"];
            }
            try
            {
                uPass=encryptString(uPass);
                using(OperationContextScope scope=new OperationContextScope
                (client.InnerChannel))
                {
                    MessageHeader user=MessageHeader.CreateHeader("uName",
                    "MySpace", uName);
                    MessageHeader pass=MessageHeader.CreateHeader("uPass",
                    "MySpace", uPass);
                    OperationContext.Current.OutgoingMessageHeaders.Add
                    (user);
                    OperationContext.Current.OutgoingMessageHeaders.Add
                    (pass);
                    client.deleteTestAsync(IDS, IDS);
                }
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
}
```

在删除完成后更新客户端的试题数据集，即编写 deleteTestCompleted 函数如下：

```
void client_deleteTestCompleted(object sender,
WCF.deleteTestCompletedEventArgs e)
{
    if(e.Error==null)
    {
        List<int>IDS=(List<int>)e.UserSatate;
        for(int ID in IDS)
        {
            DataRow row=dt.AsEnumerable().FirstOrDefault(x=>(int)x
            ["ID"]==ID);
            if(row !=null)
            {
                row.Delete();
                //从数据集中删除这一行,但是不影响别的行的状态
                row.AcceptChanges();
            }
        }
    }
    else showMsg(e.Error.Message);
}
```

4.4 试题记录更新

4.4.1 案例展示

客户端单击“获取试题”按钮就可以从服务器获取随机的 10 个题目，它们显示在一个 DataGrid 中，管理员可以更改任何一个试题的标题、内容、答案，其中答案通过嵌入在 DataGrid 中的下拉列表框来更改，更改完毕单击“更新试题”按钮就把选择的这个题目的所有的更新提交给服务器，服务器完成对应的更新，如图 4-21 所示。

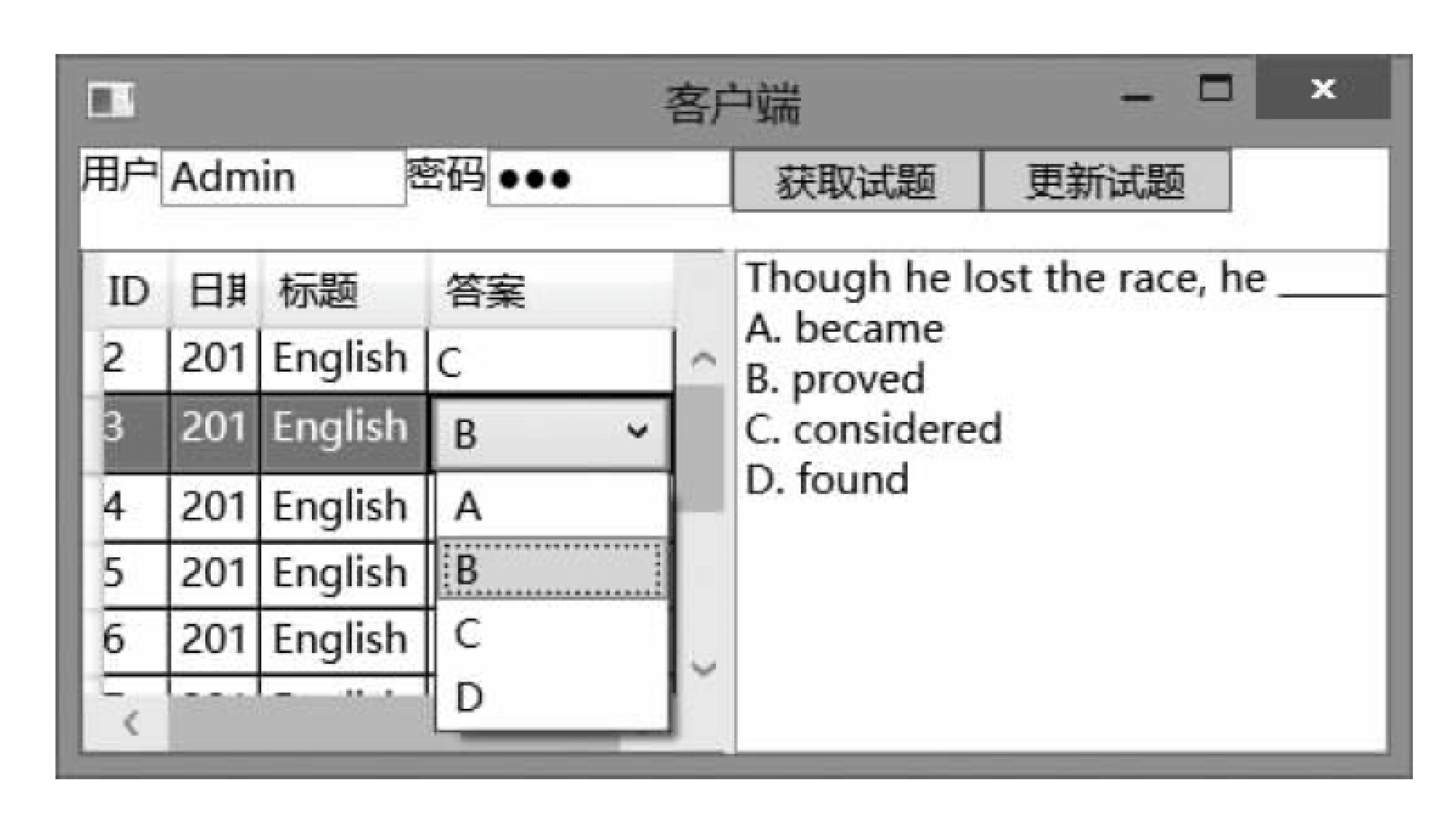


图 4-21 试题更新

4.4.2 技术要点

1. 服务器管理

在服务器的 TestService 类中增加试题修改函数 updateTest，函数以试题类 TestClass 作为参数，另外，增加 getTestDataTable 函数获取试题的数据集。所有这些操作都必须经过管理员用户身份验证，只有管理员可以进行试题的管理。

```
namespace NetTestDAL
{
    public class TestService
    {
        bool verifyAdmin(DBHelper DB, UserClass user)
        {
            if(user.uName=="Admin")
            {
                SqlParameter pName=new SqlParameter("@uName", SqlDbType.
                Char); pName.Value=user.uName;
                SqlParameter pPass=new SqlParameter("@uPass", SqlDbType.
                Char); pPass.Value=user.uPass;
                return((int)DB.getScalar("select count(*) from users where
                uName=@uName and uPass=@uPass", pName, pPass)>0);
            }
            return false;
        }
        public DataTable getTestDataTable(UserClass user)
        {
            DataTable dt=null;
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                if(verifyAdmin(DB, user))
                {
                    SqlDataAdapter adapter=DB.getAdapter("select ID,tDate,
                    tTitle,tAnswer,tText from tests order by ID");
                    dt=new DataTable("Test");
                    adapter.Fill(dt);
                }
                con.Close();
            }
            return dt;
        }
        public void updateTest(UserClass user, TestClass test)
```

```
        {
            using(SqlConnection con=new SqlConnection(DBHelper.conString))
            {
                DBHelper DB=new DBHelper(con);
                if(verifyAdmin(DB, user))
                {
                    test.tDate=DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
                    SqlParameter pDate=new SqlParameter("@tDate", SqlDbType.
                    Char); pDate.Value=test.tDate;
                    SqlParameter pTitle=new SqlParameter("@tTitle", SqlDbType.
                    Char); pTitle.Value=test.tTitle;
                    SqlParameter pText=new SqlParameter("@tText", SqlDbType.
                    Text); pText.Value=test.tText;
                    SqlParameter pAnswer = new SqlParameter ( " @ tAnswer ",
                    SqlDbType.Char); pAnswer.Value=test.tAnswer;
                    DB.executeCommand("update tests set tDate=@tDate,tTitle=
                    @tTitle,tText=@tText,tAnswer=@tAnswer where ID="+test.
                    ID.ToString(), pDate, pTitle, pText, pAnswer);
                }
                con.Close();
            }
        }
    }
```

在 NetNetTestBLL 中编写对应的管理函数，它们与 TestService 中的函数相对应，完成试题数据集的获取、试题的修改与删除等操作。

```
namespace NetTestBLL
{
    public class TestManager
    {
        public DataTable getTestDataTable(UserClass user)
        {
            TestService service=new TestService();
            return service.getTestDataTable(user);
        }
        public void updateTest(UserClass user, TestClass test)
        {
            TestService service=new TestService();
            service.updateTest(user, test);
        }
    }
}
```

2. 客户端管理

客户端采用一个 DataGrid 显示试题数据集，其中的试题标题列 tTitlet、答案列 tAnswer 是可以修改的。在答案列中修改时嵌入了一个 ComboBox 下拉列表框，编辑时答案只能在下拉列表框的选择项 A、B、C、D 中进行选择。为了实现这个功能，可以先定义一个 AsnwerList 列表类，它是 List<String>的扩展类：

```
public class AnswerClass : List<String>
{
    public AnswerClass()
    {
        Add("A"); Add("B"); Add("C"); Add("D");
    }
}
```

这个类中有 A、B、C、D 4 个选项，然后把这个类做成一个 Window 的静态资源：

```
<Window x:Class="NetTestClient.MainWindow"
    xmlns:NetTest="clr-namespace:NetTestClient" >
<Window.Resources>
    <NetTest:AnswerClass x:Key="answerList" />
</Window.Resources>
```

在 DataGrid 中定义一个<DataGridComboBoxColumn>下拉列表列：

```
<DataGridComboBoxColumn Header="答案" ItemsSource="{StaticResource
answerList}" Width="50" SelectedValueBinding="{Binding tAnswer, Mode=
TwoWay}"/>
```

这个列的数据源是一个答案列表，选择绑定数据集的 tAnswer 字段，而且 Mode 是 TwoWay，这样设置后，就可以在答案列中通过下拉列表来显示与设置试题答案了。

4.4.3 服务器程序

1. INetTestService 接口

```
[ServiceContract]
public interface INetTestService
{
    [OperationContract]
    DataTable getTestDataTable();
    [OperationContract]
    void updateTest(TestClass test);
}
```

2. NetTestService 类

```
public class NetTestService : INetTestService
{
    UserClass getUser()
    {
        String uName=OperationContext.Current.IncomingMessageHeaders.
        GetHeader<String>("uName", "MySpace");
        String uPass=OperationContext.Current.IncomingMessageHeaders.
        GetHeader<String>("uPass", "MySpace");
        return new UserClass { uName=uName, uPass=uPass };
    }
    public DataTable getTestDataTable()
    {
        TestManager manager=new TestManager();
        return manager.getTestDataTable(getUser());
    }
    public void updateTest(TestClass test)
    {
        TestManager manager=new TestManager();
        manager.updateTest(getUser(), test);
    }
}
```

4.4.4 客户端程序

运行服务器程序 NetTestServer 使得服务器处于监听状态，然后在客户端程序的解决方案资源管理器中找到 Service References 中的 WCF，右击 WCF，弹出快捷菜单，执行"更新服务引用"命令，客户端就能找到并更新服务引用。

1. 界面设计

客户端的 WPF 程序的窗体上放一个名为 uGrid 的 DataGrid 数据集显示控件显示试题数据集，放一个名称为 txtText 的 TexBox 控件显示试题内容。

```
<Window x:Class="NetTestClient.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:NetTest="clr-namespace:NetTestClient"
        Title="客户端" Height="318.817" Width="439.302" >
    <Window.Resources>
        <NetTest:AnswerClass x:Key="answerList" />
    </Window.Resources>
    <Grid>
```

```
<Grid.RowDefinitions>
    <RowDefinition Height="30"/>
    <RowDefinition Height="*" />
</Grid.RowDefinitions>
<StackPanel Orientation="Horizontal" Grid.Row="0">
    <TextBlock HorizontalAlignment="Left"  TextWrapping="Wrap"
    Text="用户" VerticalAlignment="Top" />
    <TextBox x:Name="txtUser" Text="Admin" HorizontalAlignment=
    "Left" VerticalAlignment="Top" Width="74"/>
    <TextBlock HorizontalAlignment="Left" Text="密码"
    VerticalAlignment="Top" />
    <PasswordBox x:Name="txtPass" Password="123"
    HorizontalAlignment="Left" VerticalAlignment="Top"  Width="74"/>
    <Button x:Name="btGetTest" Content="获取试题"
    HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
    Click="btGetTest_Click"/>
    <Button x:Name="btUpdateTest" Content="更新试题"
    HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
    Click="btUpdateTest_Click"/>
</StackPanel>
<Grid Grid.Row="1">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="1*" />
    </Grid.ColumnDefinitions>
    <DataGrid x:Name="uGrid" Grid.Column="0" AutoGenerateColumns=
    "False" SelectionMode="Single" SelectionChanged="uGrid_
    SelectionChanged" CanUserDeleteRows="False" CanUserAddRows=
    "False" CanUserReorderColumns="False" CanUserResizeRows="False">
        <DataGrid.Columns>
            <DataGridTextColumn Binding="{Binding ID}" Header="ID"
            Width="50" IsReadOnly="True"/>
            <DataGridTextColumn Binding="{Binding tDate}" Header="日
            期" Width="100" IsReadOnly="True" />
            <DataGridTextColumn Binding="{Binding tTitle}" Header=
            "标题" Width="100" />
            <DataGridComboBoxColumn Header="答案" ItemsSource=
            "{StaticResource answerList}" Width="50"
            SelectedValueBinding="{Binding tAnswer,Mode=TwoWay}"/>
        </DataGrid.Columns>
    </DataGrid>
    <GridSplitter Width="3" />
    <TextBox x:Name="txtText" Grid.Column="1" Text="{Binding tText, Mode
    =TwoWay}" VerticalScrollBarVisibility="Auto" />
```

```
        </Grid>
    </Grid>
</Window>
```

2. 程序设计

在程序中先定义一个 AnswerList 类，在<DataGridComboBoxColumn>列中要用到该类，这个类可以放在 MainWindow. xaml. cs 文件中，程序代码如下：

```
namespace NetTestClient
{
    public class AnswerClass : List<String>
    {
        public AnswerClass()
        {
            Add("A"); Add("B"); Add("C"); Add("D");
        }
    }
    public partial class MainWindow : Window
    {
        WCF.NetTestServiceClient client;
        String url="http://localhost:8888/NetTestService/";
        DataTable dt;
        DataView dv;
        public MainWindow()
        {
            InitializeComponent();
            //建立 client 对象
            client=new WCF.NetTestServiceClient();
            //设置异步函数
            client.getTestDataTableCompleted +=
            client_getTestDataTableCompleted;
            client.updateTestCompleted+=client_updateTestCompleted;
            //设置访问的服务器地址
            client.Endpoint.Address=new System.ServiceModel.EndpointAddress
            (new Uri(url, UriKind.Absolute));
        }
        void client_updateTestCompleted(object sender,
        System.ComponentModel.AsyncCompletedEventArgs e)
        {
            if(e.Error==null)
            {
                dt.AcceptChanges();
            }
```

```
    else showMsg(e.Error.Message);
}
void client_getTestDataTableCompleted(object sender,
WCF.getTestDataTableCompletedEventArgs e)
{
    if(e.Error==null)
    {
        dt=e.Result;
        dv=dt.DefaultView;
        uGrid.ItemsSource=dv;
    }
    else showMsg(e.Error.Message);
}
void showMsg(String s)
{
    MessageBox.Show(s, "Information", MessageBoxButton.OK);
}
String encryptString(String s)
{
    MD5 md5=new MD5CryptoServiceProvider();
    byte[] buf=Encoding.UTF8.GetBytes(s);
    buf=md5.ComputeHash(buf);
    s="";
    foreach (byte x in buf) s=s+x.ToString("X2");
    return s;
}
private void btGetTest_Click(object sender, RoutedEventArgs e)
{
    //获取试题列表
    String uName=txtUser.Text.Trim();
    String uPass=txtPass.Password.Trim();
    if(uName !="" && uPass !="")
    {
        try
        {
            uPass=encryptString(uPass);
            using(OperationContextScope scope=new OperationContextScope
            (client.InnerChannel))
            {
                MessageHeader user=MessageHeader.CreateHeader
                ("uName","MySpace", uName);
                MessageHeader pass=MessageHeader.CreateHeader
                ("uPass", "MySpace", uPass);
                OperationContext.Current.OutgoingMessageHeaders.Add
```

```
                (user);
                OperationContext.Current.OutgoingMessageHeaders.Add
                (pass);
                client.getTestDataTableAsync();
            }
        }
        catch(Exception exp) { showMsg(exp.Message); }
    }
}
private void uGrid_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if(uGrid.SelectedIndex>=0)
    {
        txtText.DataContext=dv[uGrid.SelectedIndex];
    }
    else txtText.DataContext=null;
}
private void btUpdateTest_Click(object sender, RoutedEventArgs e)
{
    if(uGrid.SelectedIndex>=0)
    {
        int index=uGrid.SelectedIndex;
        WCF.TestClass test=new WCF.TestClass
        {
            ID=(int)dv[index]["ID"],
            tTitle=dv[index]["tTitle"].ToString(),
            tAnswer=dv[index]["tAnswer"].ToString(),
            tText=dv[index]["tText"].ToString()
        };
        String uName=txtUser.Text.Trim();
        String uPass=txtPass.Password.Trim();
        if(uName !="" && uPass !="")
        {
            try
            {
                uPass=encryptString(uPass);
                using(OperationContextScope scope=new
                OperationContextScope(client.InnerChannel))
                {
                    MessageHeader user=MessageHeader.CreateHeader
                    ("uName", "MySpace", uName);
                    MessageHeader pass=MessageHeader.CreateHeader
                    ("uPass", "MySpace", uPass);
```

```
                    OperationContext.Current.OutgoingMessageHeaders.
                    Add(user);
                    OperationContext.Current.OutgoingMessageHeaders.
                    Add(pass);
                    client.updateTestAsync(test, test.ID);
                }
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
  }
 }
}
```

4.4.5 拓展训练

试题实际上可以批量更新，即先修改任意多条记录，然后单击“更新试题”按钮后把修改过的记录一次性提交服务器完成批量更新。

1. 服务器批量更新

在服务器端更改 updateTest，使得它的参数是一个 List<TestClass>对象，它包含了所有要更新的试题对象，把要更新的试题逐个更新，函数如下：

```
public void updateTest(UserClass user, List<TestClass>tests)
{
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyAdmin(DB, user))
        {
            foreach (TestClass test in tests)
            {
                test.tDate=DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
                SqlParameter pDate=new SqlParameter("@tDate", SqlDbType.
                Char); pDate.Value=test.tDate;
                SqlParameter pTitle=new SqlParameter("@tTitle", SqlDbType.
                Char); pTitle.Value=test.tTitle;
                SqlParameter pText=new SqlParameter("@tText", SqlDbType.
                Text); pText.Value=test.tText;
                SqlParameter pAnswer=new SqlParameter("@tAnswer", SqlDbType.
                Char); pAnswer.Value=test.tAnswer;
                DB.executeCommand("update tests set tDate=@tDate,tTitle=
                @tTitle,tText=@tText,tAnswer=@tAnswer where ID="+test.ID.
```

```
                ToString(), pDate, pTitle, pText, pAnswer);
            }
        }
        con.Close();
    }
}
```

同时修改 TestManager、INetTestService、NetTestService 对应的函数 updateTest 的参数，使得它们也变成一样的 List<TestClass>对象。

2. 客户端批量更新

在 DataTable 的数据集中，每行都是 DataRow 对象，这个对象有一个 RowState 属性，初始时 RowState 值保持为 Unchanged。如果一行被修改过，那么状态就变成 Modified。在更新之前可以在客户端对数据集的多行记录做修改，任何一行的修改都会把这一行的 RowState 设置为 Modified。因此在批量提交之前只要找到所有被修改过的行，把这些行的试题组成一个 List<WCF.TestClass>列表对象提交给服务器，就可以完成批量更新。根据这个方法设计更新函数如下：

```
private void btUpdateTest_Click(object sender, RoutedEventArgs e)
{
    String uName=txtUser.Text.Trim();
    String uPass=txtPass.Password.Trim();
    if(uName !="" && uPass !="")
    {
        uPass=encryptString(uPass);
        List<WCF.TestClass>tests=new List<WCF.TestClass>();
        foreach(DataRow row in dt.Rows)
        {
            if(row.RowState==DataRowState.Modified)
            {
                tests.Add(new WCF.TestClass { ID=(int)row["ID"], tTitle=
                row["tTitle"].ToString(), tAnswer=row["tAnswer"].ToString(),
                tText=row["tText"].ToString() });
            }
        }
        if(tests.Count>0)
        {
            try
            {
                using(OperationContextScope scope=new OperationContextScope
                (client.InnerChannel))
                {
                    MessageHeader user=MessageHeader.CreateHeader("uName",
```

```
                    "MySpace", uName);
                    MessageHeader pass=MessageHeader.CreateHeader("uPass",
                    "MySpace", uPass);
                    OperationContext.Current.OutgoingMessageHeaders.Add(user);
                    OperationContext.Current.OutgoingMessageHeaders.Add(pass);
                    client.updateTestListAsync(tests.ToArray());
                }
            }
            catch(Exception exp) { showMsg(exp.Message); }
        }
    }
}
```

在批量更新完成后，这个数据集必须调用 AcceptChanges 函数，使得数据集的每行状态再次恢复到 Unchanged 状态。

4.5 用户试题练习

4.5.1 案例展示

客户端单击“获取试题”按钮后会从服务器端获取一份随机抽取的试题，这个试题数据集展现在一个 DataGrid 页面上，通过下拉列表选择每个题目的答案完成答题，完成后单击“提交答案”按钮就会显示出成绩，同时将成绩提交服务器保存，如图 4-22 所示。



图 4-22 试题测试

4.5.2 技术要点

1. 成绩表

用户进行一次测试练习后的成绩被上传到数据库的 marks 表进行存储，这个表有一个 ID 自动增长字段作为关键字，uName 是用户名称，mDate 为存储日期，mValue 是成绩。表格的 SQL 命令如下：

```
create table marks
(
    ID int identity(1,1) primary key,
    uName varchar(32) foreign key references users(uName) on delete cascade,
    mDate varchar(32),
    mValue int
)
```

为了保持数据的完整性，uName 必须外键参照 users 表中的 uName 字段。

2. 服务器管理

在 TestService.cs 中设计函数来进行用户练习试题与成绩的管理。用户在获取试题时要进行身份验证，设计一个验证函数 verifyUser 用来验证用户的身份，这个函数与管理员验证函数 verifyAdmin 十分类似，只有注册过的用户才可以获取试题。

```
bool verifyUser(DBHelper DB, UserClass user)
{
    //验证用户身份
    SqlParameter pName=new SqlParameter("@uName", SqlDbType.Char); pName.
    Value=user.uName;
    SqlParameter pPass=new SqlParameter("@uPass", SqlDbType.Char); pPass.
    Value=user.uPass;
    return((int)DB.getScalar("select count(*) from users where uName=@uName
    and uPass=@uPass", pName, pPass)>0);
}
```

由于服务器端的试题很多，为了实现测试练习的随机性，用户在练习时，程序每次随机地从试题表 tests 中抽取 10 个题目组成练习试卷。在 SQL Server 数据库中通过 NEWID 函数可以为每条记录生成一个随机数，如果按这些随机数来排序，每次提取排序后的前 10 个题目，就可以得到随机的 10 个题目了，关键的 SQL 语句如下：

```
select top 10 * from tests order by NEWID()
```

根据这个方法编写 getUserTestDataTable 函数，它有一个 UserClass 参数，验证用户身份后，从 tests 中随机抽取 10 个题目组成一个数据集 DataTable，函数如下：

```
public DataTable getUserTestDataTable(UserClass user)
{
    Random rnd=new Random();
    DataTable dt=null;
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyUser(DB, user))
        {
```

```
                //NEWID()用来产生随机顺序
                SqlDataAdapter adapter=DB.getAdapter("select top 10 * from tests
                order by NEWID()");
                dt=new DataTable("Test");
                adapter.Fill(dt);
            }
            con.Close();
        }
        return dt;
    }
```

用户完成答题后提交试题，程序比对用户答案与标准答案，计算出成绩 mValue，通过 setUserMark 函数把成绩存储到 marks 表中，这个函数如下：

```
    public int setUserMark(UserClass user, int mValue)
    {
        int flag=-1;
        using(SqlConnection con=new SqlConnection(DBHelper.conString))
        {
            DBHelper DB=new DBHelper(con);
            if(verifyUser(DB, user))
            {
                SqlParameter pName=new SqlParameter("@uName", SqlDbType.Char);
                pName.Value=user.uName;
                SqlParameter pDate=new SqlParameter("@mDate", SqlDbType.Char);
                pDate.Value=DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
                SqlParameter pValue=new SqlParameter ("@mValue", SqlDbType.
                Char); pValue.Value=mValue;
                if(DB.executeCommand("insert into marks (uName, mDate, mValue)
                values (@uName,@mdate,@mValue)", pName, pDate, pValue)>0) flag=
                mValue;
            }
            con.Close();
        }
        return flag;
    }
```

在设计好 NetTestDAL 的函数后，接下来在 NetTestBLL 的 TestManager 中设计类似的 getUserTestDataTable 函数与 setUserMark 函数，它们与 NetTestService 的同名函数对应，这里不再赘述。

3. 客户端管理

客户端获取试题数据集 dt 后，在 dt 中增加两个数据列：一个是 sNo，代表题目的序号；另一个是 uAnswer，代表用户的答案。因此在获取数据集后执行下列程序：

```
dt.Columns.Add("tNo");
dt.Columns.Add("uAnswer");
for(int i=0;i<dt.Rows.Count;i++)
{
    dt.Rows[i]["tNo"]=(i+1).ToString();
    dt.Rows[i]["uAnswer"]="";
}
dt.AcceptChanges();
```

这样,在界面中设计一个 DataGrid 控件来显示 sNo 与 uAnswer 列数据,用户答案初始为空白。

4.5.3 服务器程序

1. INetTestService 接口

```
[ServiceContract]
public interface INetTestService
{
    [OperationContract]
    DataTable getUserTestDataTable();
    [OperationContract]
    int setUserMark(int mValue);
}
```

2. NetTestService 类

```
public class NetTestService : INetTestService
{
    UserClass getUser()
    {
        String uName=OperationContext.Current.IncomingMessageHeaders.
        GetHeader<String>("uName", "MySpace");
        String uPass=OperationContext.Current.IncomingMessageHeaders.
        GetHeader<String>("uPass", "MySpace");
        return  new UserClass { uName=uName, uPass=uPass };
    }
    public DataTable getUserTestDataTable()
    {
        TestManager manager=new TestManager();
        return manager.getUserTestDataTable(getUser());
    }
    public int setUserMark(int mValue)
    {
        TestManager manager=new TestManager();
```

```
            manager.setUserMark(getUser(), mValue);
        }
    }
```

4.5.4 客户端程序

首先运行服务器程序 NetTestServer 使得服务器处于监听状态，然后在客户端程序的解决方案资源管理器中找到 Service References 中的 WCF，右击 WCF，弹出快捷菜单，执行“更新服务引用”命令，客户端就能找到并更新服务引用。

1. 界面设计

客户端的界面主要是一个 DataGrid，它有两个列，一个绑定题目序号，另一个绑定用户答案，主要 XAML 代码如下：

```
<Window x:Class="NetTestClient.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:NetTest="clr-namespace:NetTestClient"
        Title="客户端" Height="255.617" Width="439.302" >
    <Window.Resources>
        <NetTest:AnswerClass x:Key="answerList" />
    </Window.Resources>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="30"/>
            <RowDefinition Height=" * " />
        </Grid.RowDefinitions>
        <StackPanel Orientation="Horizontal" Grid.Row="0">
            <TextBlock HorizontalAlignment="Left"  TextWrapping="Wrap" Text
            ="用户" VerticalAlignment="Top" />
            <TextBox x:Name="txtUser" Text="xxx" HorizontalAlignment="Left"
            VerticalAlignment="Top" Width="74"/>
            <TextBlock HorizontalAlignment="Left" Text="密码"
            VerticalAlignment="Top" />
            <PasswordBox x:Name="txtPass" Password="123"  Horizontal-
            Alignment="Left" VerticalAlignment="Top"  Width="74"/>
            <Button x:Name="btGetTest" Content="获取试题"
            HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
            Click="btGetTest_Click"/>
            <Button x:Name="btHandleTest" Content="提交答案"
            HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
            Click="btHandleTest_Click"/>
            <TextBlock x:Name="msg" Text="" Foreground="Red" />
```

```
        </StackPanel>
        <Grid Grid.Row="1">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="120" />
                <ColumnDefinition Width="1*" />
            </Grid.ColumnDefinitions>
            <DataGrid x:Name="uGrid" Grid.Column="0" AutoGenerateColumns=
           "False"   SelectionMode =" Single "  SelectionChanged =" uGrid _
            SelectionChanged" CanUserDeleteRows =" False " CanUserAddRows =
           "False" CanUserReorderColumns="False" CanUserResizeRows="False">
                <DataGrid.Columns>
                    <DataGridTextColumn Binding="{Binding tNo}" Header="序号"
                    Width="40" IsReadOnly="True"/>
                    <DataGridComboBoxColumn Header="答案"
                    ItemsSource="{StaticResource answerList}" Width="80"
                    SelectedValueBinding="{Binding uAnswer,Mode=TwoWay}"/>
                </DataGrid.Columns>
            </DataGrid>
            <GridSplitter Width="3" />
            <TextBox x:Name="txtText" Grid.Column="1"
            Text="{Binding tText, Mode=TwoWay}"
            VerticalScrollBarVisibility="Auto" />
        </Grid>
    </Grid>
</Window>
```

2. 程序设计

```
namespace NetTestClient
{
    public class AnswerClass : List<String>
    {
        public AnswerClass()
        {
            Add("A"); Add("B"); Add("C"); Add("D");
        }
    }
    public partial class MainWindow : Window
    {
        WCF.NetTestServiceClient client;
        String url="http://localhost:8888/NetTestService/";
        DataTable dt;
        DataView dv;
        public MainWindow()
```

```
{
    InitializeComponent();
    //建立 client 对象
    client=new WCF.NetTestServiceClient();
    //设置异步函数
    client.getUserTestDataTableCompleted+=
    client_getUserTestDataTableCompleted;
    //设置访问的服务器地址
    client.Endpoint.Address=new System.ServiceModel.EndpointAddress
    (new Uri(url, UriKind.Absolute));
    btHandleTest.IsEnabled=false;
}
void client_getUserTestDataTableCompleted(object sender,
WCF.getUserTestDataTableCompletedEventArgs e)
{
    if(e.Error==null)
    {
        dt=e.Result;
        dt.Columns.Add("tNo");
        dt.Columns.Add("uAnswer");
        for(int i=0;i<dt.Rows.Count;i++)
        {
            dt.Rows[i]["tNo"]=(i+1).ToString();
            dt.Rows[i]["uAnswer"]="";
        }
        dt.AcceptChanges();
        dv=dt.DefaultView;
        uGrid.ItemsSource=dv;
        btGetTest.IsEnabled=false;
        btHandleTest.IsEnabled=true;
    }
    else showMsg(e.Error.Message);
}
void showMsg(String s)
{
    MessageBox.Show(s, "Information", MessageBoxButton.OK);
}
String encryptString(String s)
{
    MD5 md5=new MD5CryptoServiceProvider();
    byte[] buf=Encoding.UTF8.GetBytes(s);
    buf=md5.ComputeHash(buf);
    s="";
    foreach (byte x in buf) s=s+x.ToString("X2");
```

```
        return s;
    }
    private void uGrid_SelectionChanged(object sender,
    SelectionChangedEventArgs e)
    {
        if(uGrid.SelectedIndex>=0)
        {
            txtText.DataContext=dv[uGrid.SelectedIndex];
        }
        else txtText.DataContext=null;
    }
    private void btHandleTest_Click(object sender, RoutedEventArgs e)
    {
        int mValue=0;
        foreach(DataRow row in dt.Rows)
        {
            if(row["tAnswer"].ToString()==row["uAnswer"].ToString())++
           mValue;
        }
        msg.Text="成绩:"+mValue.ToString();
        btHandleTest.IsEnabled=false;
        btGetTest.IsEnabled=true;
        String uName=txtUser.Text.Trim();
        String uPass=txtPass.Password.Trim();
        if(uName !="" && uPass !="")
        {
            try
            {
                uPass=encryptString(uPass);
                using(OperationContextScope scope=new OperationContext-
                Scope(client.InnerChannel))
                {
                    MessageHeader user=MessageHeader.CreateHeader
                    ("uName", "MySpace", uName);
                    MessageHeader pass=MessageHeader.CreateHeader
                    ("uPass", "MySpace", uPass);
                    OperationContext.Current.OutgoingMessageHeaders.Add
                    (user);
                    OperationContext.Current.OutgoingMessageHeaders.Add
                    (pass);
                    client.setUserMarkAsync(mValue);
                }
            }
            catch(Exception exp) { showMsg(exp.Message); }
```

```
            }
        }
        private void btGetTest_Click(object sender, RoutedEventArgs e)
        {
            //获取试题列表
            String uName=txtUser.Text.Trim();
            String uPass=txtPass.Password.Trim();
            if(uName !="" && uPass !="")
            {
                try
                {
                    uPass=encryptString(uPass);
                    using(OperationContextScope scope=new OperationContext-
                    Scope(client.InnerChannel))
                    {
                        MessageHeader user=MessageHeader.CreateHeader
                        ("uName", "MySpace", uName);
                        MessageHeader pass=MessageHeader.CreateHeader
                        ("uPass", "MySpace", uPass);
                        OperationContext.Current.OutgoingMessageHeaders.Add
                        (user);
                        OperationContext.Current.OutgoingMessageHeaders.Add
                        (pass);
                        client.getUserTestDataTableAsync();
                    }
                }
                catch(Exception exp) { showMsg(exp.Message); }
            }
        }
    }
}
```

4.5.5 拓展训练

用户提交的成绩存储在服务器端的数据库中,可以设计一个功能从客户端来管理成绩。

1. 服务器管理

首先在服务器端的 TestService.cs 中设计一个获取成绩的函数 getMarkDataTable,它接收一个用户 UserClass 参数,验证用户。如果为管理员 Admin,就获取全部成绩;如果是普通用户,就获取用户自己的成绩。

```
public DataTable getMarkDataTable(UserClass user)
```

```
{
    Random rnd=new Random();
    DataTable dt=null;
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyAdmin(DB, user))
        {
            SqlDataAdapter adapter=DB.getAdapter("select * from marks order
            by ID");
            dt=new DataTable("Mark");
            adapter.Fill(dt);
        }
        else if(verifyUser(DB,user))
        {
            SqlParameter pName=new SqlParameter("@uName", SqlDbType.Char);
            pName.Value=user.uName;
            SqlDataAdapter adapter=DB.getAdapter("select * from marks where
            uName=@uName order by ID",pName);
            dt=new DataTable("Mark");
            adapter.Fill(dt);
        }
        con.Close();
    }
    return dt;
}
```

另外设计一个删除成绩的函数 deleteMarkList，它接收一个成绩记录的 ID 号列表 List<int>参数，其中包含了所有要删除的成绩记录的 ID 号。

```
public void deleteMarkList(UserClass user, List<int>mID)
{
    using(SqlConnection con=new SqlConnection(DBHelper.conString))
    {
        DBHelper DB=new DBHelper(con);
        if(verifyUser(DB, user))
        {
            foreach(int ID in mID)
            {
                DB.executeCommand("delete from marks where ID="+ID.ToString());
            }
        }
        con.Close();
    }
}
```

接下来在 TestManager、INetTestService 及 NetTestService 中编写对应的 getMarkDataTable 函数与 deleteMarkList 函数。

2. 客户端管理

客户端设计一个成绩管理窗体 MarkWindow，其中窗体中用一个 DataGrid 控件来显示成绩，DataGrid 支持多行选择，一次可以选择多行要删除的成绩记录。同时设计一个“删除成绩”按钮，单击该按钮后删除选择的成绩记录。

1）界面设计

```
<Window x:Class="NetTestClient.MarkWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="成绩记录" x:Name="markWindow" Height="300" Width="300"
        Loaded="markWindow_Loaded" WindowStartupLocation="CenterScreen"
        Closing="markWindow_Closing">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="30" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Button Grid.Row="0" x:Name="deleteMark"
        Content="删除成绩" HorizontalAlignment="Center" VerticalAlignment=
       "Center" Click="deleteMark_Click" />
        <DataGrid x:Name="mGrid" Grid.Row="1" AutoGenerateColumns="False"
        IsReadOnly="True" SelectionMode="Extended"  >
            <DataGrid.Columns>
                <DataGridTextColumn Binding="{Binding uName}" Header="用户"
                Width="100" />
                <DataGridTextColumn Binding="{Binding mDate}" Header="日期"
                Width="100" />
                <DataGridTextColumn Binding="{Binding mValue}" Header="成绩"
                Width="100" />
            </DataGrid.Columns>
        </DataGrid>
    </Grid>
</Window>
```

2）程序设计

```
public partial class MarkWindow : Window
{
    public MainWindow mainWnd;
    public WCF.UserClass currentUser;
    //服务器地址
```

```
public String url;
//客户端对象
private WCF.NetTestServiceClient client;
//成绩数据集与视图
private DataTable dt;
private DataView dv;
public MarkWindow()
{
    InitializeComponent();
    //建立 client 对象
    client=new WCF.NetTestServiceClient();
    //设置异步函数
    client.getMarkDataTableCompleted+=client_getMarkData-
    TableCompleted;
    client.deleteMarkListCompleted+=client_deleteMarkListCompleted;
}
void client_deleteMarkListCompleted(object sender,
System.ComponentModel.AsyncCompletedEventArgs e)
{
    if(e.Error==null)
    {
        List<int>mID=(List<int>)e.UserState;
        foreach(int ID in mID)
        {
            DataRow row=dt.AsEnumerable().FirstOrDefault(x=>(int)x
            ["ID"]==ID);
            if(row !=null) { row.Delete(); row.AcceptChanges(); }
        }
        dt.AcceptChanges();
    }
    else mainWnd.showMsg(e.Error.Message);
}
void client_getMarkDataTableCompleted(object sender,
WCF.getMarkDataTableCompletedEventArgs e)
{
    if(e.Error==null)
    {
        if(e.Result !=null)
        {
            dt=e.Result;
            dv=dt.DefaultView;
            mGrid.ItemsSource=dv;
        }
        else mGrid.ItemsSource=null;
```

```
        }
        else mainWnd.showMsg(e.Error.Message);
    }
    private void markWindow_Loaded(object sender, RoutedEventArgs e)
    {
        //窗体启动时获取成绩记录
        //设置访问的服务器地址
        client.Endpoint.Address=new System.ServiceModel.EndpointAddress
        (new Uri(url, UriKind.Absolute));
        try
        {
            using(OperationContextScope scope=new OperationContextScope
            (client.InnerChannel))
            {
                MessageHeader user=MessageHeader.CreateHeader("uName",
                "MySpace", currentUser.uName);
                MessageHeader pass=MessageHeader.CreateHeader("uPass",
                "MySpace", currentUser.uPass);
                OperationContext.Current.OutgoingMessageHeaders.Add(user);
                OperationContext.Current.OutgoingMessageHeaders.Add(pass);
                client.getMarkDataTableAsync();
            }
        }
        catch(Exception exp) { mainWnd.showMsg(exp.Message); }
    }
    private void markWindow_Closing(object sender, System.ComponentModel.
    CancelEventArgs e)
    {
        mainWnd.Show();
    }
    private void deleteMark_Click(object sender, RoutedEventArgs e)
    {
        //删除选择的成绩记录
        if(mGrid.SelectedItems.Count>0)
        {
            List<int>mID=new List<int>();
            for (int i=0; i<mGrid.SelectedItems.Count; i++)
            {
                DataRowView row=(DataRowView)mGrid.SelectedItems[i];
                mID.Add((int)row["ID"]);
            }
            if(mID.Count>0)
            {
                if(mainWnd.confirm("确实要删除选择成绩记录?"))
```

```
                {
                    try
                    {
                        using(OperationContextScope scope=new
                        OperationContextScope(client.InnerChannel))
                        {
                            MessageHeader user=MessageHeader.CreateHeader
                            ("uName", "MySpace", currentUser.uName);
                            MessageHeader pass=MessageHeader.CreateHeader
                            ("uPass", "MySpace", currentUser.uPass);
                            OperationContext.Current.OutgoingMessageHeaders.
                            Add(user);
                            OperationContext.Current.OutgoingMessageHeaders.
                            Add(pass);
                            client.deleteMarkListAsync(mID.ToArray(), mID);
                        }
                    }
                    catch(Exception exp) { mainWnd.showMsg(exp.Message); }
                }
            }
        }
        else mainWnd.showMsg("请选择要删除的记录!");
    }
}
```

在练习窗体中增加一个“成绩管理”按钮，单击这个按钮后打开成绩窗体，对成绩进行管理，同时练习窗体隐藏不可见，当成绩窗体关闭后再次显示练习窗体。成绩窗体启动函数如下：

```
private void btMark_Click(object sender, RoutedEventArgs e)
{
    MarkWindow dlg=new MarkWindow();
    dlg.mainWnd=this;
    dlg.url=url;
    dlg.currentUser=currentUser;
    dlg.Show();
    this.Hide();
}
```

其中 mainWnd、url、currentUser 是 MarkWindow 窗体(成绩窗体)中的变量，mainWnd 是主窗体对象，currentUser 为用户对象，url 为服务器地址，这些信息在主窗体中就已经确定，所以直接传递给 MarkWindow。

4.6 试题练习程序的实现

4.6.1 技术要点

现在可以把本项目前面几节的内容连贯起来，完成这个试题练习程序的总体设计。

在测试练习时，用户单击“获取试题”后得到一个试题的数据集，为了更好地展现这些试题，下面设计一个带滚动条的 StackPanel 容器 spTest：

```
<ScrollViewer HorizontalScrollBarVisibility="Auto" VerticalScrollBarVisibility
="Auto">
    <ScrollViewer.Content>
        <StackPanel Orientation="Vertical" x:Name="spTest" />
    </ScrollViewer.Content>
</ScrollViewer>
```

它包含在一个 ScrollViewer 控件中，当 spTest 中的内容比较多时就自动出现水平与垂直的滚动条。在 spTest 中动态增加所要的控件来展示试题，spTest 包含的所有子控件都在它的 Children 属性中。

每个试题有 3 个部分，先在 spTest 中增加一个 TextBlock 用来显示试题的内容，再增加一个 ComboBox 供用户选择答案。随后增加一个 TextBlock 存储标准答案，这个 TextBlock 的名称设置为 tbAns，这个答案控件在练习时不可见，只有在完成练习进行答案比对时才显示。最后增加一个空白的 TextBlock 用来作为两个试题的分隔区。设计 getTestPaper 函数完成试题数据集在 spTest 中的显示。主要的代码如下：

```
void getTestPaper()
{
    spTest.Children.Clear();
    for(int i=0;i<dv.Count;i++)
    {
        TextBlock tb=new TextBlock { TextWrapping=TextWrapping.NoWrap, Text
        =(i+1).ToString()+". "+dv[i]["tText"].ToString() };
        spTest.Children.Add(tb);
        StackPanel sp=new StackPanel();
        ComboBox cb= new ComboBox {Name ="cbAns"+i.ToString(), Width = 50,
        HorizontalAlignment=HorizontalAlignment.Left };
        cb.Items.Add("A");
        cb.Items.Add("B");
        cb.Items.Add("C");
        cb.Items.Add("D");
        spTest.Children.Add(cb);
        tb=new TextBlock {Name="tbAns"+i.ToString(),Text=dv[i]["tAnswer"].
        ToString(),Visibility=Visibility.Hidden,Foreground=Brushes.Red };
```

```
            spTest.Children.Add(tb);
            tb=new TextBlock { Text="" };
            spTest.Children.Add(tb);
        }
    }
```

在用户提交答案时查找 spTest.Children 中的每个控件元素，每个元素都是最基础的 UIElement 类元素。如果这个元素是 ComboBox，就取出用户答案；如果是 TextBlock 而且名称以 Ans 开始，就是答案控件，从中取出标准答案，对比用户答案与标准答案，就可以计算出成绩 mValue。主要的代码如下：

```
List<String> sAns=new List<string>();      //标准答案
List<String> tAns=new List<string>();      //用户答案
for(int i=0;i<spTest.Children.Count;i++)
{
    UIElement elem=spTest.Children[i];
    if(elem is ComboBox)
    {
        ComboBox cb=(ComboBox)elem;
        if(cb.SelectedItem!=null) sAns.Add(cb.SelectedItem.ToString());
        else sAns.Add("");
    }
    else if(elem is TextBlock)
    {
        TextBlock tb=(TextBlock)elem;
        if(tb.Name.StartsWith("tbAns")) { tAns.Add(tb.Text); tb.Visibility=
       Visibility.Visible; }
    }
}
//计算成绩
int mValue=0;
for(int i=0;i<sAns.Count;i++)
    if(tAns[i]==sAns[i])++mValue;
```

4.6.2 服务器程序

服务器由 NetTestDAL 数据访问层、NetTestBLL 业务逻辑层、NetTestModel 数据模型层、NetTestServer 应用程序层组成，如图 4-23 所示。

1. NetTestDAL

该层主要包含数据库的基础操作类 DBHelper 与用户注册登录的管理类 UserService，这两个类的功能与前面项目的类似。试题数据的管理类 TestService 中主要的函数如表 4-1 所示。

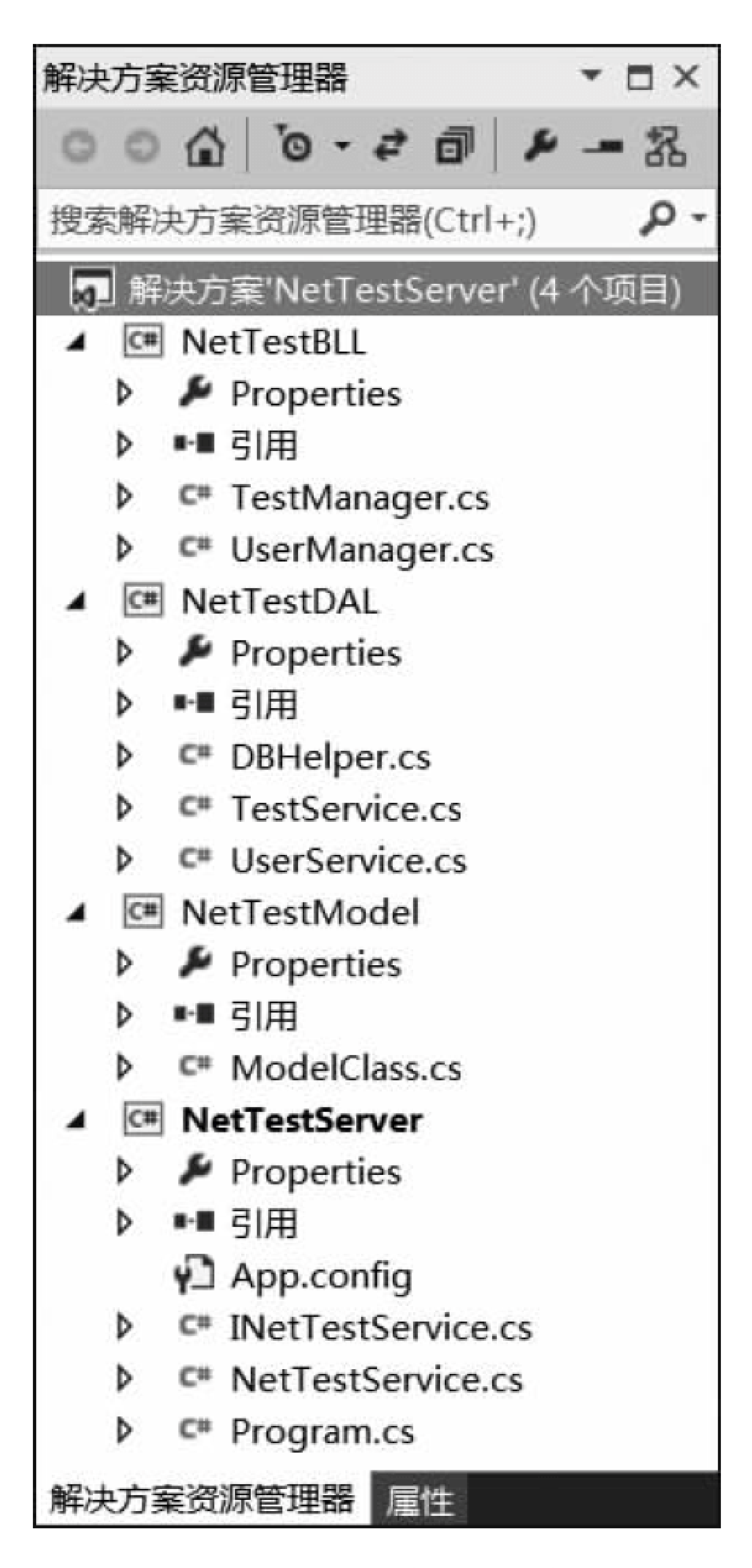


图 4-23 服务器结构

表 4-1 TestService 的主要函数

函数名称	说明
verifyAdmin(UserClass)	判断用户是否为管理员
verifyUser(UserClass)	判断用户是否为注册用户
addTest(UserClass,ref TestClass):int	增加一个试题
updateTest(UserClass,ref TestClass)	更新一个试题
updateTestList(UserClass,List＜TestClass＞)	批量更新试题
deleteTest(UserClass,TestClass)	删除一个试题
getTestDataTable(UserClass)	获取试题数据集
getUserTestDataTable(UserClass)	获取随机的测试数据集
setUserMark(UserClass,int)	设置一次测试的成绩
getMarkDataTable(UserClass)	获取成绩数据集
deleteMarkList(UserClass,List＜int＞)	批量删除成绩

2. NetTestBLL

该层的逻辑比较简单，因此它包含的文件与函数几乎是 NetTestDAL 对应文件与函数的再写。

3. NetTestModel

这个模型中主要包含用户类 UserClass、试题类 TestClass，内容如下：

```
namespace NetTestModel
{
    public class UserClass
    {
        public String uName { get; set; }
        public String uPass { get; set; }
    }
    public class TestClass
    {
        public int ID { get; set; }
        public String tDate { get; set; }
        public String tTitle { get; set; }
        public String tAnswer { get; set; }
        public String tText { get; set; }
    }
}
```

4. WCF 服务

NetTestService 类实现 INetTestService 接口的各个函数，这些函数如下：

```
public class NetTestService : INetTestService
{
    public String login(UserClass user)
    {
        UserManager manager=new UserManager();
        return manager.login(user);
    }
    UserClass getUser()
    {
        String uName=OperationContext.Current.IncomingMessageHeaders.
        GetHeader<String>("uName", "MySpace");
        String uPass=OperationContext.Current.IncomingMessageHeaders.
        GetHeader<String>("uPass", "MySpace");
        return new UserClass { uName=uName, uPass=uPass };
    }
```

```
public TestClass addTest(TestClass test)
{
     TestManager manager=new TestManager();
    manager.addTest(getUser(), ref test);
    return test;
}
public DataTable getTestDataTable()
{
    TestManager manager=new TestManager();
    return manager.getTestDataTable(getUser());
}
public bool deleteTest(TestClass test)
{
    TestManager manager=new TestManager();
    return manager.deleteTest(getUser(),test);
}
public TestClass updateTest(TestClass test)
{
    TestManager manager=new TestManager();
    manager.updateTest(getUser(), ref test);
    return test;
}
public void updateTestList(List<TestClass>tests)
{
    TestManager manager=new TestManager();
    manager.updateTestList(getUser(), tests);
}
public DataTable getUserTestDataTable()
{
    TestManager manager=new TestManager();
    return manager.getUserTestDataTable(getUser());
}
public int setUserMark(int mValue)
{
    TestManager manager=new TestManager();
    return manager.setUserMark(getUser(),mValue);
}
public DataTable getMarkDataTable()
{
    TestManager manager=new TestManager();
    return manager.getMarkDataTable(getUser());
}
public void deleteMarkList(List<int>mID)
{
```

```
            TestManager manager=new TestManager();
            manager.deleteMarkList(getUser(),mID);
        }
    }
```

其中除了 getUser 函数不是 INetTestService 接口定义的函数外，其他全部是 INetTestService 接口中定义的函数，它们都是 WCF 的服务器接口函数。

4.6.3 客户端程序

客户端由一个主窗体 MainWindow、一个试题管理窗体 AdminWindow、一个用户测试练习窗体 TestWindow、一个成绩管理窗体 MarkWindow 组成，如图 4-24 所示。



图 4-24　客户端结构

1. MainWindow

MainWindow(主窗体)负责用户注册与登录，根据是 Admin 管理员用户还是普通用户设置对应的功能按钮的有效性。单击“试题管理”“成绩管理”“测试练习”按钮分别打开 AdminWindow、MarkWindow、TestWindow 窗体。例如，打开 TestWindow 窗体程序如下：

```
private void btTest_Click(object sender, RoutedEventArgs e)
{
    TestWindow dlg=new TestWindow();
    dlg.mainWnd=this;
    dlg.url=url;
    dlg.currentUser=currentUser;
    dlg.Show();
    this.Hide();
}
```

其中mainWnd、url、currentUser是TestWindow窗体中的变量，mainWnd是主窗体对象，currentUser为登录的用户对象，url为服务器地址，这些信息在主窗体中用户登录后就已经确定，所以直接传递给TestWindow。

2. TestWindow

这个窗体是用户进行试题练习的窗体，其中主要的特点是用了一个带滚动条的StackPanel来展现试题。

1）TestWindow界面

```
<Window x:Name="testWindow" x:Class="NetTestClient.TestWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="测试练习" Height="428" Width="493.6"
        Closing="testWindow_Closing" Loaded="testWindow_Loaded"
        WindowStartupLocation="CenterScreen">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="30"/>
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <StackPanel Orientation="Horizontal" Grid.Row="0">
            <Button x:Name="btGetTest" Content="获取试题"
            HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
            Click="btGetTest_Click"/>
            <Button x:Name="btHandleTest" Content="提交答案"
            HorizontalAlignment="Left" VerticalAlignment="Top" Width="75"
            Click="btHandleTest_Click"/>
            <TextBlock x:Name="msg" Foreground="Red" />
        </StackPanel>
        <Grid Grid.Row="1" Margin="10,10,10,10">
            <ScrollViewer HorizontalScrollBarVisibility="Auto"
            VerticalScrollBarVisibility="Auto">
                <ScrollViewer.Content>
                    <StackPanel Orientation="Vertical" x:Name="spTest" />
                </ScrollViewer.Content>
            </ScrollViewer>
        </Grid>
    </Grid>
</Window>
```

2）TestWindow程序

```
public partial class TestWindow : Window
{
    public MainWindow mainWnd;
```

```
public WCF.UserClass currentUser;
//服务器地址
public String url;
//客户端对象
private WCF.NetTestServiceClient client;
//数据集与视图
private DataTable dt;
private DataView dv;
public TestWindow()
{
    InitializeComponent();
    //建立 client 对象
    client=new WCF.NetTestServiceClient();
    //设置异步函数
    client.getUserTestDataTableCompleted+=
    client_getUserTestDataTableCompleted;
    client.setUserMarkCompleted+=client_setUserMarkCompleted;
    //设置访问的服务器地址
    btHandleTest.IsEnabled=false;
}
void client_setUserMarkCompleted(object sender,
WCF.setUserMarkCompletedEventArgs e)
{
    if(e.Error==null)
    {
        msg.Text="成绩:"+e.Result.ToString();
        btGetTest.IsEnabled=true;
        btHandleTest.IsEnabled=false;
    }
    else showMsg(e.Error.Message);
}
void client_getUserTestDataTableCompleted(object sender,
WCF.getUserTestDataTableCompletedEventArgs e)
{
    if(e.Error==null)
    {
        dt=e.Result;
        dv=dt.DefaultView;
        getTestPaper();
        msg.Text="";
        btGetTest.IsEnabled=false;
        btHandleTest.IsEnabled=true;
    }
    else showMsg(e.Error.Message);
}
void getTestPaper()
```

```
{
    spTest.Children.Clear();
    for (int i=0; i<dv.Count; i++)
    {
        TextBlock tb=new TextBlock { TextWrapping=TextWrapping.NoWrap,
        Text=(i+1).ToString()+". "+dv[i]["tText"].ToString() };
        spTest.Children.Add(tb);
        StackPanel sp=new StackPanel();
        ComboBox cb=new ComboBox { Name="cbAns"+i.ToString(), Width=50,
        HorizontalAlignment=HorizontalAlignment.Left };
        cb.Items.Add("A");
        cb.Items.Add("B");
        cb.Items.Add("C");
        cb.Items.Add("D");
        spTest.Children.Add(cb);
        tb=new TextBlock { Name="tbAns"+i.ToString(), Text=
        dv[i]["tAnswer"].ToString(), Visibility=Visibility.Hidden,
        Foreground=Brushes.Red };
        spTest.Children.Add(tb);
        tb=new TextBlock { Text="" };
        spTest.Children.Add(tb);
    }
}
void showMsg(String s)
{
    MessageBox.Show(s, "Information", MessageBoxButton.OK);
}
private void btGetTest_Click(object sender, RoutedEventArgs e)
{
    //获取试题列表
    try
    {
          using (OperationContextScope scope = new OperationContextScope
          (client.InnerChannel))
        {
            MessageHeader user=MessageHeader.CreateHeader("uName",
            "MySpace", currentUser.uName);
            MessageHeader pass=MessageHeader.CreateHeader("uPass",
            "MySpace", currentUser.uPass);
            OperationContext.Current.OutgoingMessageHeaders.Add(user);
            OperationContext.Current.OutgoingMessageHeaders.Add(pass);
            client.getUserTestDataTableAsync();
        }
    }
    catch(Exception exp) { showMsg(exp.Message); }
}
```

```
private void btHandleTest_Click(object sender, RoutedEventArgs e)
{
    List<String>sAns=new List<string>();
    List<String>tAns=new List<string>();
    for (int i=0; i<spTest.Children.Count; i++)
    {
        UIElement elem=spTest.Children[i];
        if(elem is ComboBox)
        {
            ComboBox cb=(ComboBox)elem;
            if(cb.SelectedItem !=null) sAns.Add(cb.SelectedItem.
            ToString());
            else sAns.Add("");
        }
        else if(elem is TextBlock)
        {
            TextBlock tb=(TextBlock)elem;
            if(tb.Name.StartsWith("tbAns")) { tAns.Add(tb.Text);
            tb.Visibility=Visibility.Visible; }
        }
    }
    int mValue=0;
    for (int i=0; i<sAns.Count; i++)
        if(tAns[i]==sAns[i])++mValue;
    try
    {
        using(OperationContextScope scope=new OperationContextScope
        (client.InnerChannel))
        {
            MessageHeader user=MessageHeader.CreateHeader("uName",
            "MySpace", currentUser.uName);
            MessageHeader pass=MessageHeader.CreateHeader("uPass",
            "MySpace", currentUser.uPass);
            OperationContext.Current.OutgoingMessageHeaders.Add(user);
            OperationContext.Current.OutgoingMessageHeaders.Add(pass);
            client.setUserMarkAsync(mValue);
        }
    }
    catch(Exception exp) { showMsg(exp.Message); }
}
private void testWindow_Closing(object sender, System.ComponentModel.
CancelEventArgs e)
{
    mainWnd.Show();
}
private void testWindow_Loaded(object sender, RoutedEventArgs e)
```

```
        {
            client.Endpoint.Address=new System.ServiceModel.EndpointAddress
            (new Uri(url, UriKind.Absolute));
        }
    }
```

另外，AdminWindow、MarkWindow 的界面与程序都与前面的类似，不再赘述。

4.6.4 拓展训练

这个项目的服务器是一个独立运行的控制台程序，客户端是一个 WPF 的窗体程序，实际上 WCF 的程序结构是很灵活的，服务器可以是一个独立于 IIS 的应用程序，也可以是一个 Windows 服务程序，还可以是基于 IIS 的网站服务。同时客户端可以是一个 Windows 或者 WPF 窗体程序，也可以是基于 IIS 的网页应用程序。

1. 基于 IIS 的服务器服务

如果要把服务器改成依赖 IIS 的服务程序，那么适当修改 NetTestServer 服务器项目就可以了，具体步骤如下：

(1) 打开 NetTestServer 项目，删除 NetTestServer 的控制台程序项目。

(2) 添加一个网站(例如 D:\web)项目，把网站 web 设置为启动项目，并增加 web 网站对 NetTestBLL、NetTestModel 的引用。

(3) 在 web 中添加 WCF 服务 NetTestService，得到一个 NetTestService.svc 文件，这是 IIS 中的 WCF 服务器文件。同时在 web 下建立 App_Code 文件夹，并在 App_Code 中建立 INetTestService.cs 与 NetTestService.cs 两个文件，如图 4-25 所示。

(4) INetTestService.cs 与 NetTestService.cs 文件内容与控制台程序下的对应文件几乎完全一样，不同的是不再有 NetTestServer 的命名空间，因此 INetTestService.cs 代码如下：

```
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
using NetTestModel;
using System.Data;
[ServiceContract]
public interface INetTestService
{
```

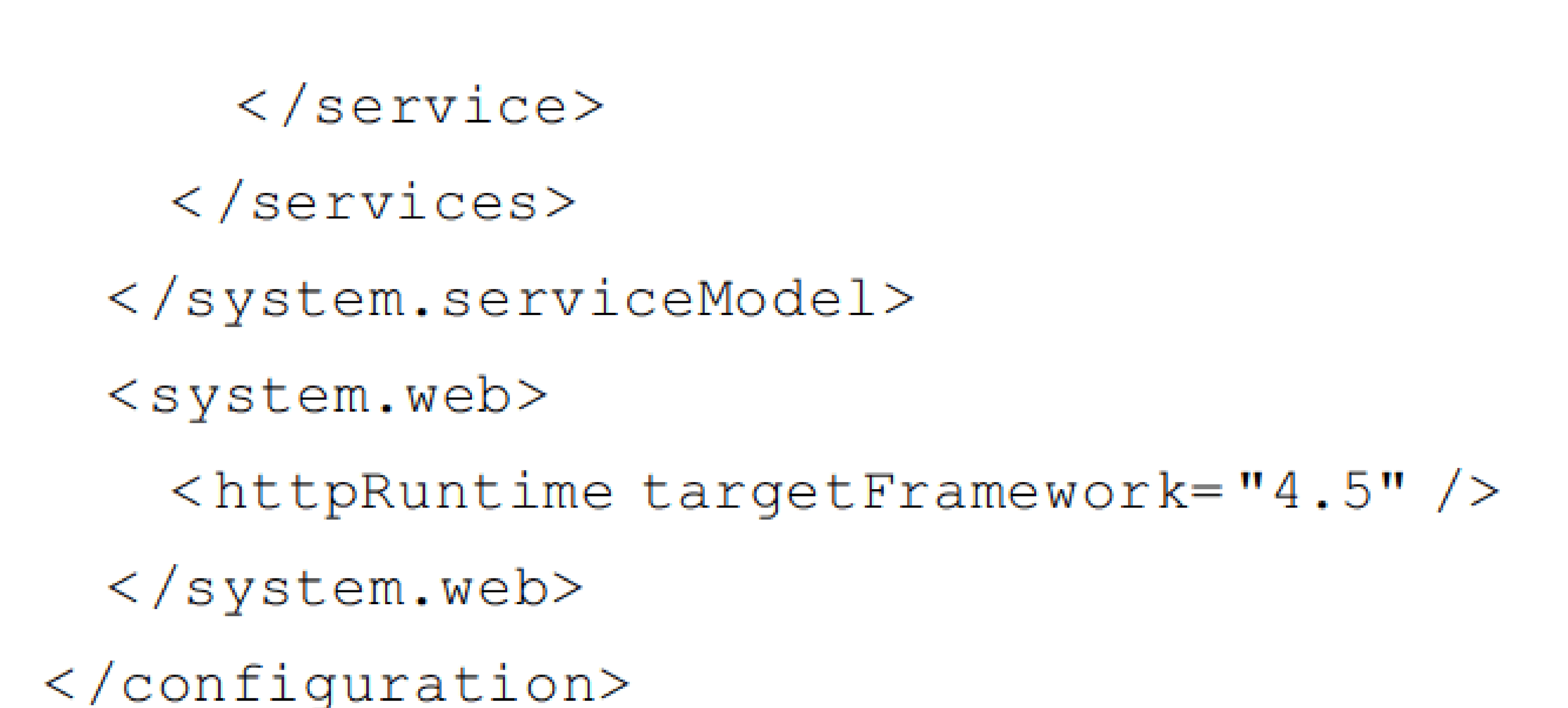


图 4-25 web 网站的 WCF 服务

```
    [OperationContract]
    String login(UserClass user);
    [OperationContract]
    TestClass addTest(TestClass test);
    [OperationContract]
    DataTable getTestDataTable();
    [OperationContract]
    bool deleteTest(TestClass test);
    [OperationContract]
    TestClass updateTest(TestClass test);
    [OperationContract]
    void updateTestList(List<TestClass>tests);
    [OperationContract]
    DataTable getUserTestDataTable();
    [OperationContract]
    int setUserMark(int mValue);
    [OperationContract]
    DataTable getMarkDataTable();
    [OperationContract]
    void deleteMarkList(List<int>mID);
}
```

(5) 修改 web.config 文件,使它变成如下的形式:

```
<?xml version="1.0"?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="NetTestBehavior">
          <serviceMetadata httpGetEnabled="true"/>
          <serviceDebug includeExceptionDetailInFaults="false"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="NetTestService" behaviorConfiguration="NetTestBehavior">
        <endpoint address="" binding="basicHttpBinding"
        contract="INetTestService">
          <identity>
            <dns value="Localhost"/>
          </identity>
        </endpoint>
        <endpoint contract="IMetadataExchange" binding="mexHttpBinding"
        address="mex"/>
```



```
      </service>
    </services>
  </system.serviceModel>
  <system.web>
    <httpRuntime targetFramework="4.5" />
  </system.web>
</configuration>
```

(6)重新生成解决方案，在 web\bin 目录下自动生成 NetTestDAL、NetTestBLL、NetTestModel 等的动态库。

(7)用浏览器浏览网址 http://localhost/web/NetTestService.svc，结果如图 4-26 所示，从结果中看到服务已经创建成功。

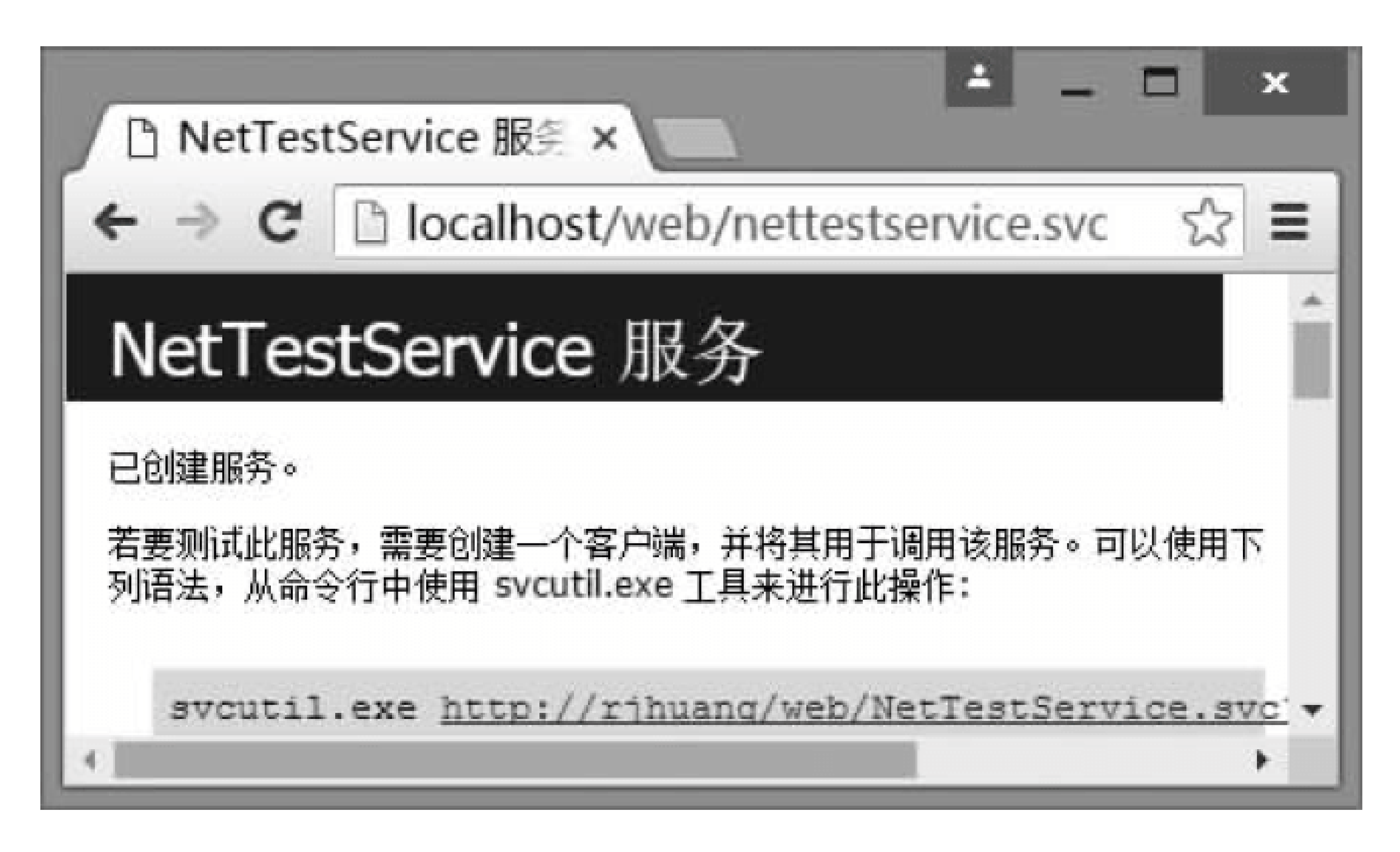


图 4-26　已创建服务

经过上面这些步骤就已经把服务器变成了基于 IIS 的服务程序。现在启动客户端程序 NetTestClient，把服务器地址改成 http://localhost/web/NetTestService.svc，单击“登录”按钮，效果与连接控制台服务器程序的完全一样，如图 4-27 所示。



图 4-27　客户端程序

2. 基于 IIS 的客户端网页

客户端也可以是一个基于 IIS 的 Web 网页，下面以登录页面为例说明网页的创建方法。

(1) 新建一个网站(例如 D:\client)，在其中添加一个网页 login.aspx，设计这个页面如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="login.aspx.cs"
Inherits="login" Async="true" %>
<body>
    <form id="form1" runat="server">
    <div>
    地址<asp:TextBox ID="txtURL" runat="server" Text="http://localhost/web/
    NetTestService.svc" Width="260"/><br />
    用户<asp:TextBox ID="txtName" runat="server" Text="xxx" /><br />
    密码<asp:TextBox ID="txtPass" runat="server" TextMode="Password" Text=
    "123" /><br />
    <asp:Button ID="btLogin" Text="登录" runat="server" OnClick="tryLogin" />
    <asp:Label ID="txtMsg" runat="server" />
    </div>
    </form>
</body>
```

(2) 在 client 网站中执行“添加服务引用”命令，在“地址”中输入服务器的地址，可以是

```
http://localhost/web/NetTestService.svc
```

或者(需要启动控制台的服务器程序 NetTestServer)

```
http://localhost:8888/NetTestService/
```

然后单击“转到”按钮就可以发现服务器的服务，如图 4-28 所示。

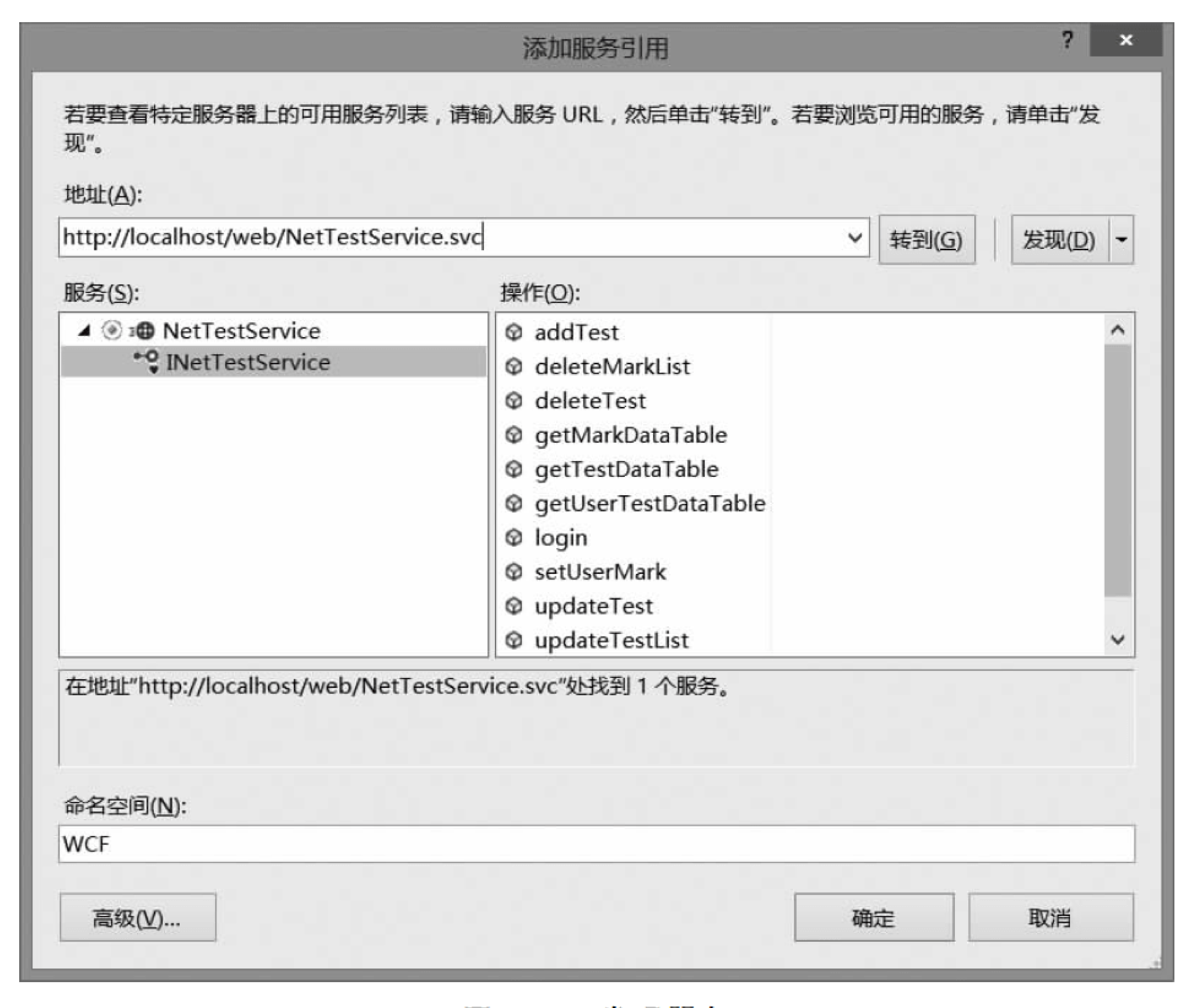


图 4-28　发现服务

在服务中设置生成异步操作，确定后在解决方案资源管理器中看到建立了一个 WCF 服务，如图 4-29 所示。



图 4-29 引用服务

(3) 编写 login.aspx 程序，使用异步操作，程序如下：

```
public partial class login: System.Web.UI.Page
{
    WCF.NetTestServiceClient client;
    protected void Page_Load(object sender, EventArgs e)
    {
        client=new WCF.NetTestServiceClient();
        client.loginCompleted+=client_loginCompleted;
    }
    void client_loginCompleted(object sender, WCF.loginCompletedEventArgs e)
    {
        if(e.Error==null) txtMsg.Text=e.Result;
        else txtMsg.Text=e.Error.Message;
    }
    String encryptString(String s)
    {
        MD5 md5=new MD5CryptoServiceProvider();
        byte[] buf=Encoding.UTF8.GetBytes(s);
        buf=md5.ComputeHash(buf);
        s="";
        foreach (byte x in buf) s=s+x.ToString("X2");
        return s;
    }
    protected void tryLogin(object sender, EventArgs e)
    {
        String url=txtURL.Text.Trim();
```

```
            String uName=txtName.Text.Trim();
            String uPass=txtPass.Text.Trim();
            try
            {
                uPass=encryptString(uPass);
                client.Endpoint.Address=new EndpointAddress(new Uri(url, UriKind.
                Absolute));
                //异步调用 login 函数
                client.loginAsync(new WCF.UserClass { uName=uName, uPass=uPass });
            }
            catch(Exception exp) { txtMsg.Text=exp.Message; }
        }
    }
```

(4) 把 d:\client 做成一个 Web 网站,用浏览器浏览,单击“登录”按钮后可以看到登录成功,如图 4-30 所示。

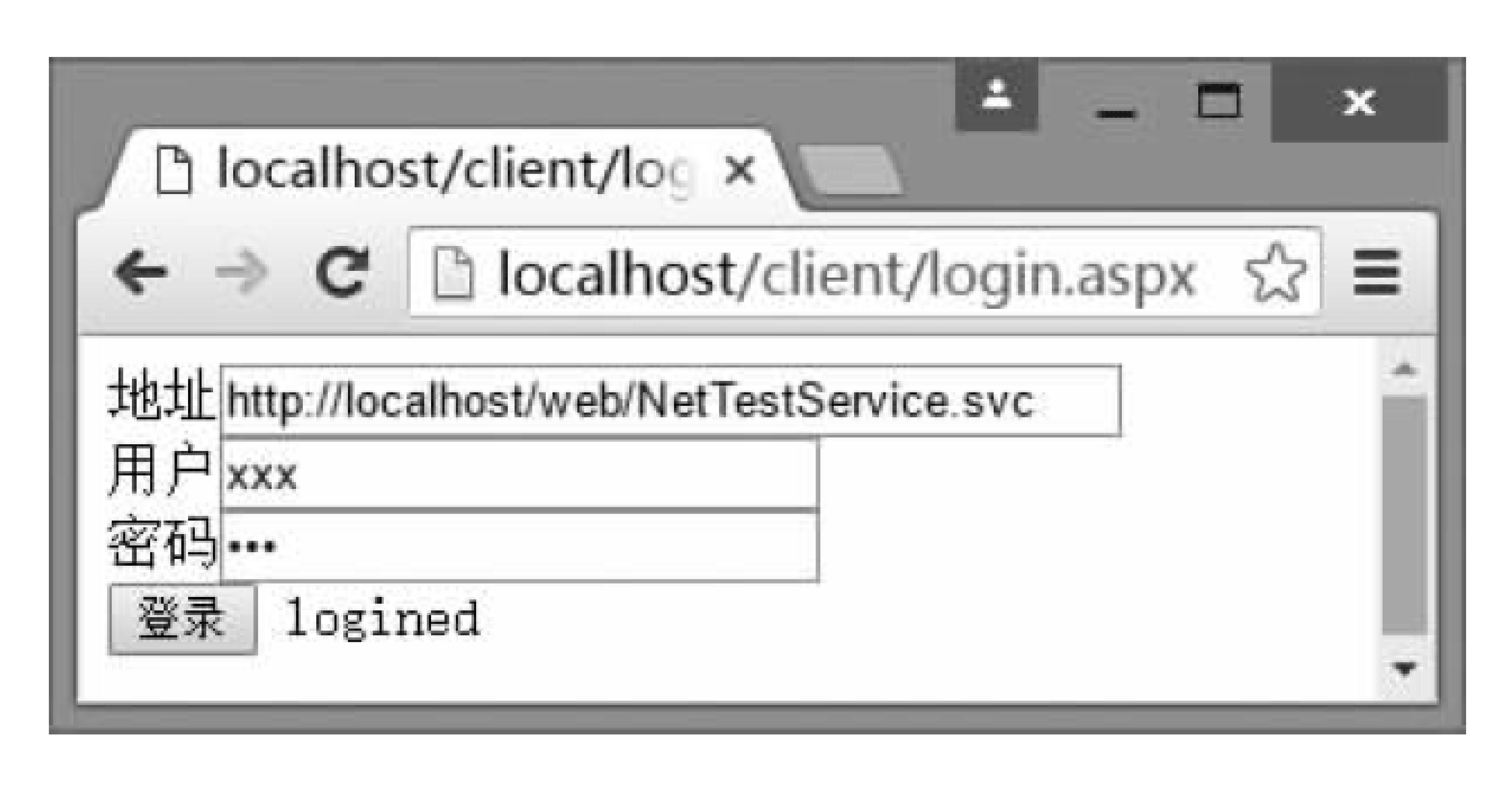


图 4-30 登录成功

很显然 login.aspx 网页与 NetTestService.svc 可以部署在两个不同的 Web 服务器上,这时一个 Web 网站的网页 login.aspx 去访问另外一个 Web 网站的 NetTestService.svc 服务,系统形成分布式的结构。

当然 login.aspx 也可以访问控制台程序的服务器,如图 4-31 所示。

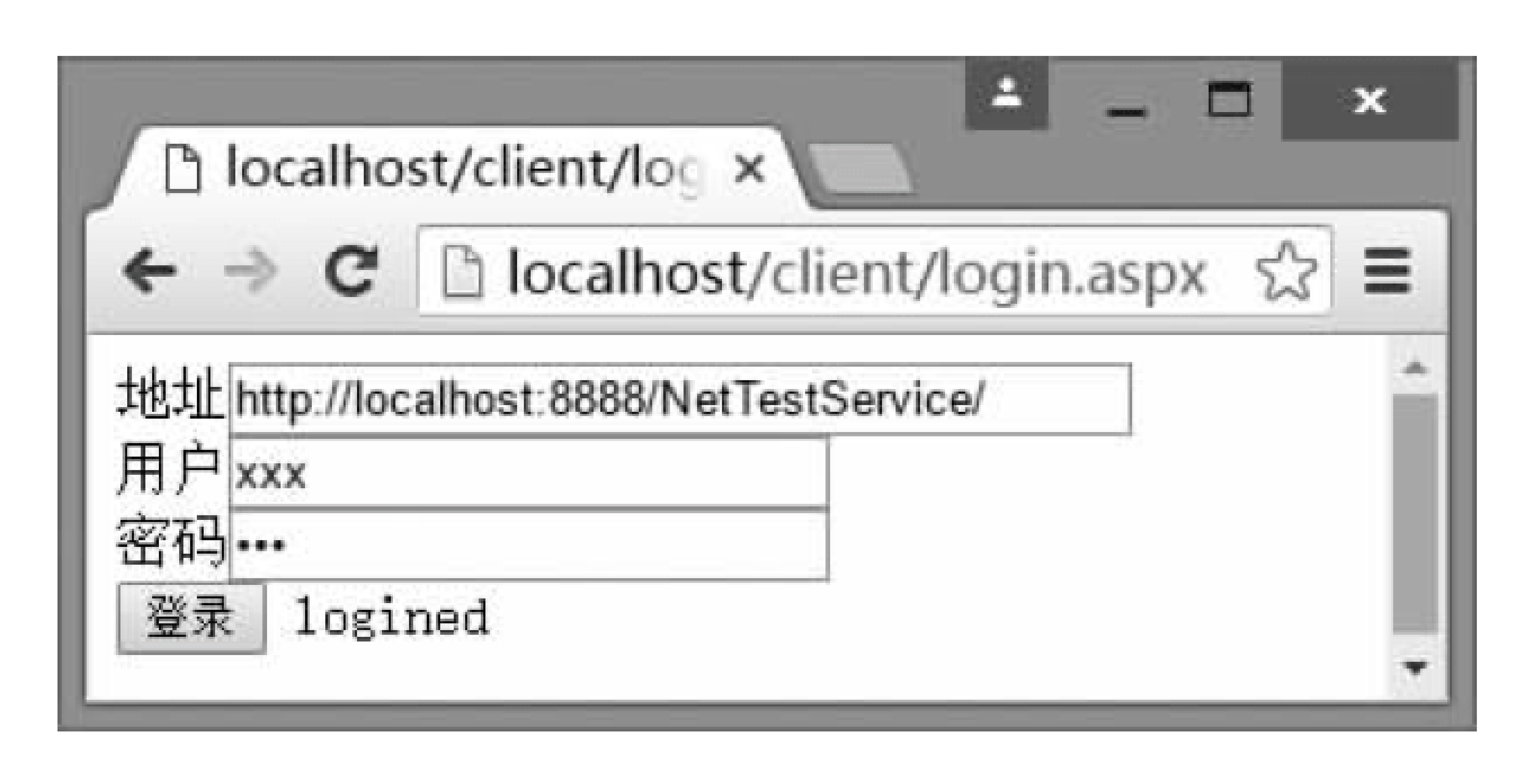


图 4-31 登录成功

有兴趣的读者可以进一步设计试题管理、用户练习、成绩管理等功能网页,就可以把客户端程序变成一个真正的网站程序了。

练　习　四

1．比较 Web Service 与 WCF 的异同。

2．编写一个 WCF 程序 server.asmx 与一个客户端程序 client，server.asmx 中包含一个接口函数 divide 实现两个数的除法运算：

```
public double divide(double x,double y)
{
    return x/y;
}
```

除法运算在 y＝0 时会出现异常，编写客户端的程序调用这个函数并捕捉可能的异常。

3．部署 WCF 服务器程序时，如何通过配置 app.config 文件来保护服务器程序，使得客户端不能发现服务器的接口函数？举例说明。

4．用 WCF 实现练习三中第 3 题的服务器图像浏览程序，其中 WCF 的服务器程序设计成基于 IIS 的 svc 程序。

5．用 WCF 编写一个网盘管理程序，实现网盘的功能：

(1) 服务器为每个注册用户分配一个文件夹作为该用户的根文件夹。

(2) 用户在客户端登录后，可以查看自己根文件夹下的所有文件及子文件夹，用户可以在根文件夹中建立任意级别的子文件夹。

(3) 用户可以把本地文件上传到任意一个指定的文件夹中，也可以下载任何一个文件到本地磁盘。

(4) 用户可以管理自己的文件与文件夹，例如，更改文件与文件夹的名称，删除文件与空的文件夹，移动和复制文件到另外一个文件夹。

6．用 WCF 编写项目二 2.4 节的图片共享程序。